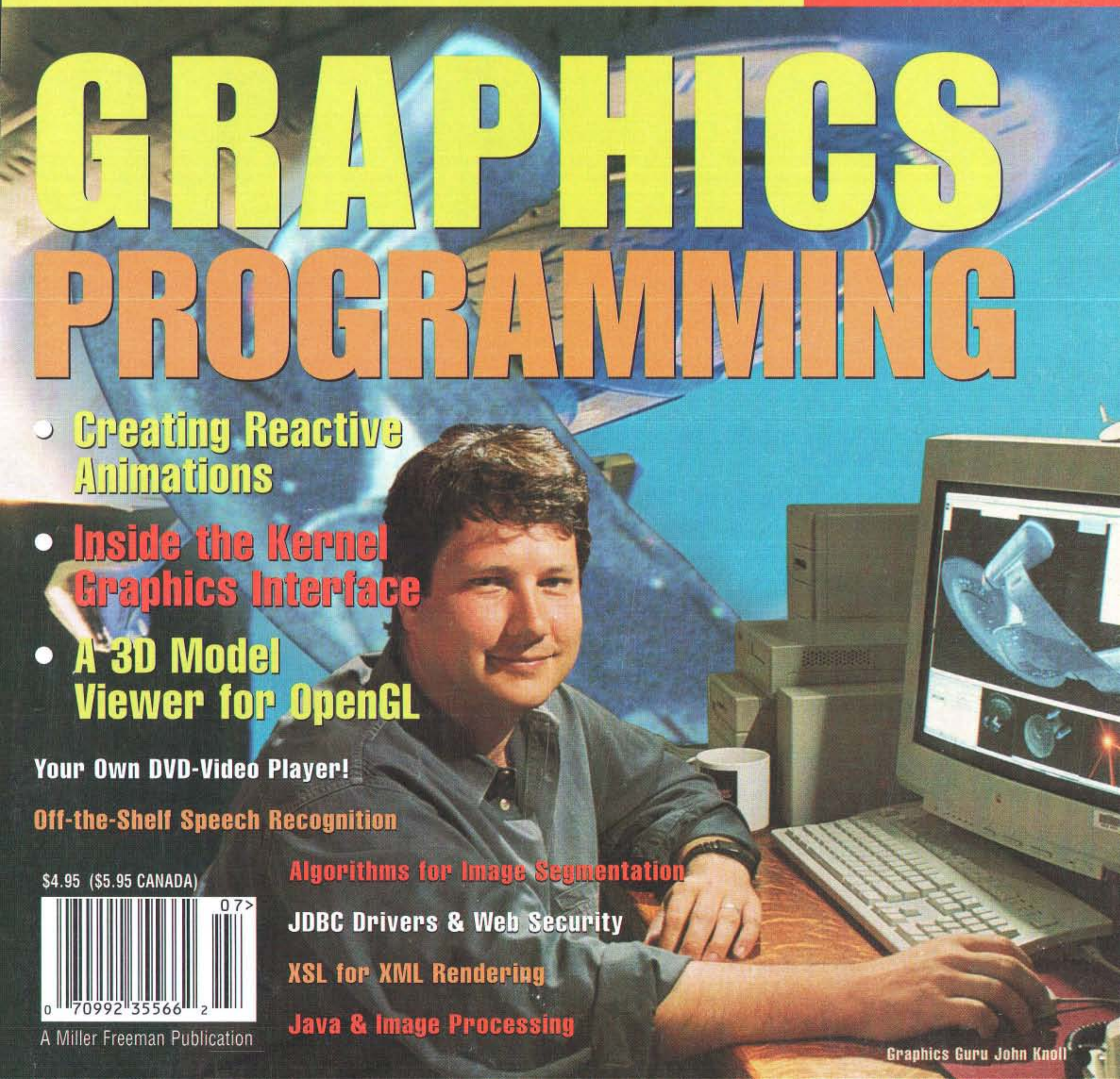
John Knoll: Industrial Light & Magic's Graphics Magician

#287 JULY 1998

Dr. Dobb's JOURNAL

SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

http://www.ddj.com



GRAPHICS PROGRAMMING

- Creating Reactive Animations
- Inside the Kernel Graphics Interface
- A 3D Model Viewer for OpenGL

Your Own DVD-Video Player!

Off-the-Shelf Speech Recognition

Algorithms for Image Segmentation

JDBC Drivers & Web Security

XSL for XML Rendering

Java & Image Processing

$4.95 ($5.95 CANADA)

0 70992 35566 2 07>

A Miller Freeman Publication

Graphics Guru John Knoll

Dr. Dobb's JOURNAL
SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

CONT

FEATURES

COMPOSING REACTIVE ANIMATIONS 18
by Conal Elliott
Fran, short for "functional reactive animation," is a high-level vocabulary that lets you describe the essential nature of an animated model, while omitting details of presentation.

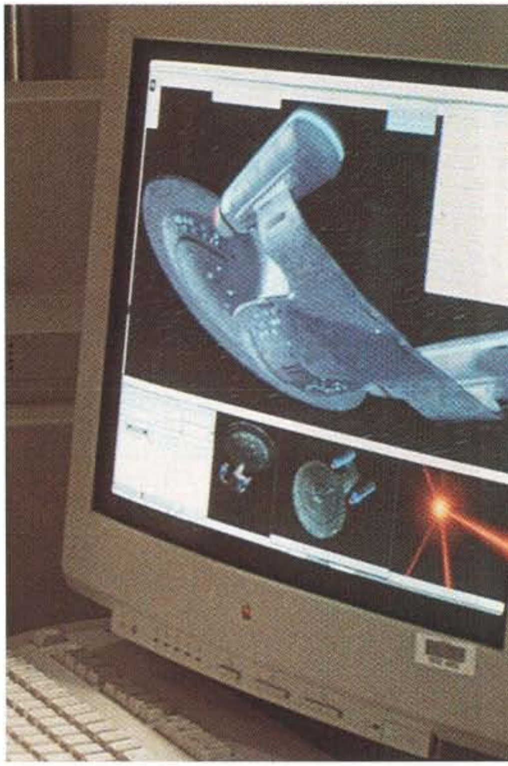
A CONVERSATION WITH JOHN KNOLL 34
by Thomas "Rick" Tewell
As a visual-effects supervisor for Industrial Light & Magic, John Knoll lives on the bleeding-edge of computer graphics. With his brother Tom, he also created the PhotoShop image-processing software.

A WINDOWS 3D MODEL VIEWER FOR OPENGL 44
by Jawed Karim
Combining Win32 with OpenGL can lead to some impressive 3D graphics. Jawed presents a model viewer for use with OpenGL on Windows 95/NT.

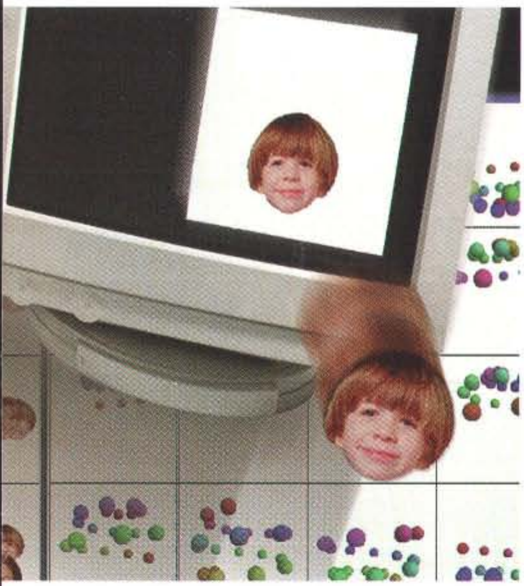
THE KERNEL GRAPHICS INTERFACE 50
by Andreas Beck
The General Graphics Interface (GGI) project brings safe, fast, and portable graphics to a variety of platforms and operating systems. Andreas describes KGI, the kernel-level component of the Linux version of GGI.

AFFINE TEXTURE MAPPING 58
by André LaMothe
Affine texture mapping is fundamental to many forms of 3D rendering, including light interpolation and other sampling type operations.

INSIDE DVD 64
by Linden deCarmo
Although DVDs physically resemble CD-ROMs, they store up to 25 times more data. Linden focuses on the DVD-Video specification, and presents a DVD-Video player.



Photography by Sean Casey courtesy of Industrial Light & Magic.



EMBEDDED SYSTEMS

68HC05-BASED PERIPHERAL DEVICES: PART II 72
by Derrick B. Forte and Hai T. Nguyen
In this two-part article, our authors design a Windows 95-based Caller ID peripheral device built around Motorola's MC68HC(7)05P9 microcontroller. This month, they present the software.

INTERNET PROGRAMMING

RENDERING XML DOCUMENTS USING XSL 82
by Sean McGrath
Responsibility for rendering XML belongs to the eXtensible Style Language (XSL) Standard. Sean presents an overview of XSL and illustrates how it can be used with MSXSL, Microsoft's XSL implementation.

ENTS

JULY 1998
VOLUME 23, ISSUE 7

PROGRAMMER'S TOOLCHEST

EXAMINING THE DRAGON SPEECH-RECOGNITION SYSTEM 86
by Al Williams
Al uses Visual Basic 5 and Dragon Systems' DragonXTools toolkit to build a voice-activated autodialer. Since the custom controls are ActiveX controls, however, you can use most any language.

JDBC DRIVERS AND WEB SECURITY 90
by Mukul Sood
Although security is not officially part of the Java Database Connectivity (JDBC) specification, JDBC driver vendors are beginning to offer security features such as encryption and authentication.

COLUMNS

PROGRAMMING PARADIGMS 101
by Michael Swaine
Michael ventures into the land of the lizards before adding his two cents to LEO lore. He then introduces a new "Paradigms Past" feature.

C PROGRAMMING 105
by Al Stevens
Al continues his journey into the mysteries of Windows CE—and the Windows CE Developers Conference has given him a lot to think about.

JAVA Q&A 109
by Aaron Michael Cohen
The Java API defines an abstract imaging model that can be used to display and manipulate both static images and sequences of images. Aaron examines this powerful yet flexible model.

ALGORITHM ALLEY 115
by Lee Kamentsky
Many image-analysis tasks must first separate the image into clearly defined regions. Lee's algorithm performs such a separation and presents the results in a fashion amenable to further study.

DR. ECCO'S OMNIHEURIST CORNER 123
by Dennis E. Shasha
Dr. Ecco and his sidekick Liane help the military avoid going from the frying pan into lines of fire in this month's episode.

PROGRAMMER'S BOOKSHELF 127
by Gregory V. Wilson
Greg looks at a bevy of books this month, including *Software Visualization, C/C++ Software Quality Tools, Perl: The Programmer's Companion, Effective Perl Programming, Perl 5 Interactive Course,* and *Software Runaways: Monumental Software Disasters.*

FORUM

EDITORIAL 6
by Jonathan Erickson

LETTERS 10
by you

NEWS & VIEWS 16
by the DDJ *staff*

OF INTEREST 130
by Eugene Eric Kim

SWAINE'S FLAMES 136
by Michael Swaine

RESOURCE CENTER

As a service to our readers, source code (and related files), back-referenced articles, and relevant links are available electronically at this month's online Table of Contents at http://www.ddj.com/. Source code is also available via anonymous FTP from ftp.ddj.com (199.125.85.76), the DDJ Forum on CompuServe (type GO DDJ), and DDJ Online (650-358-8857, 14.4 kbps, 8-N-1). Source-code diskettes can be ordered ($14.95, California residents add sales tax) by mail, fax (650-358-9749), or phone (650-655-4100 x5701). Letters to the editor and article proposals/submissions should be mailed or faxed to the *DDJ* office or sent electronically to editors@ddj.com. Author guidelines are available at http://www.ddj.com/. Send inquiries or requests to *Dr. Dobb's Journal*, 411 Borel Ave., San Mateo, CA 94402. For subscription questions (including change of address), call 800-456-1215 (U.S. and Canada); other countries, call 303-678-0439 or fax 303-661-1885. E-mail subscription questions to 71572.341@compuserve.com or write to *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80322-6188.

NEXT MONTH

August brings our annual C++ programming issue.

Dr. Dobb's JOURNAL

SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

PUBLISHER *Peter Westerman*

EDITORIAL

EDITOR-IN-CHIEF *Jonathan Erickson*
MANAGING EDITOR *Deirdre Blake*
MANAGING EDITOR, DIGITAL MEDIA *Kevin Carlson*
SENIOR TECHNICAL EDITOR *Tim Kientzle*
TECHNICAL EDITOR *Eugene Eric Kim*
SENIOR PRODUCTION EDITOR *Monica E. Berg*
ASSOCIATE MANAGING EDITOR *Amy W. Wong*
ART DIRECTOR *Margaret A. Anderson*
CONTRIBUTING EDITORS *Al Stevens, Tom Genereaux, Scot Wingo, George Shepherd, Robert Collins, Cliff Berg, Steve Alexander, David Betz, Bruce Schneier, Mark Russinovich, Bryce Cogswell, Ray Duncan, Jack Woehr, Jon Bentley, Dennis Shasha*
EDITOR-AT-LARGE *Michael Swaine*
PRODUCTION MANAGER *Denise Denis*

CIRCULATION

DIRECTOR OF CIRCULATION *Jerry Okabe*
GROUP CIRCULATION MANAGER *Michael Poplardo*

MARKETING/ADVERTISING

SALES DIRECTOR, EAST *Brenner Fuller*
SALES DIRECTOR, WEST *Paul Miller*
MARKETING MANAGER *Holly Vessichelli*
PUBLISHER'S ASSISTANT *Jodie Medeiros*
PRODUCTION COORDINATOR *Michael Calderon*
ACCOUNT MANAGERS see page 128
GRAPHIC DESIGNER *Carey Perez*

MILLER FREEMAN INC.

CEO *Tony Tillin*
CHAIRMAN *Marshall W. Freeman*
PRESIDENT *Donald A Pazour*
SENIOR VICE PRESIDENT/CFO *Warren "Andy" Ambrose*
SENIOR VICE PRESIDENTS *H. Ted Bahr, David Nussbaum, Darrell Denny, Wini D. Ragus, Galen A. Poss*
VICE PRESIDENT/SOFTWARE DIVISION *Regina Starr Ridley*
VICE PRESIDENT/PRODUCTION *Andrew A. Mickus*
VICE PRESIDENT/CIRCULATION *Jerry Okabe*
GROUP PUBLISHER *Peter Hutchinson*

DR. DOBB'S JOURNAL **ISSN 1044-789X** is published monthly by Miller Freeman, Inc., 600 Harrison Street, San Francisco, CA 94107; 415-905-2200. Periodicals Postage Paid at San Francisco and at additional mailing offices.

ARTICLE SUBMISSIONS: Send manuscript and disk (with article, listings, and letter to the editor) to *DDJ* Submissions, 411 Borel Ave., San Mateo, CA 94402-3522.

***DDJ* ON COMPUSERVE:** Type GO DDJ.

***DDJ* ON THE INTERNET:** editors@ddj.com or http://www.ddj.com/.

SUBSCRIPTION: $34.95 for 1 year; $69.90 for 2 years. International orders must be prepaid. Payment may be made via Mastercard, Visa, or American Express; or via U.S. funds drawn on a U.S. bank. Canada and Mexico: $45.00 per year. All other foreign: $70.00 per year.

FOR SUBSCRIPTION QUESTIONS, change of address, and orders, call toll-free **800-456-1215** (U.S. and Canada). For all other countries, call **303-678-0439** or fax **303-661-1885**. On CompuServe, the customer-service number is 71572,341. Or write, *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80322-6188.

BACK ISSUES may be purchased for $9.00 per copy (includes shipping and handling). For issue availability, call **800-444-4881** in the U.S. and Canada. All other countries call **785-838-7500** or fax **785-841-2624**. Send e-mail to **orders@mfi.com.** Back issue orders must be prepaid. Please send payment to *Dr. Dobb's Journal*, 1601 W. 23rd Street, Suite 200, Lawrence, KS 66046-2700.

POSTMASTER: Send address changes to *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80328-6188. **ISSN 1044-789X. GST (Canada) # R124771239.**

Canada Post International Publications Mail Product (Canadian Distribution) **Sales Agreement No. 0548677.**

FOREIGN NEWSSTAND DISTRIBUTOR: Worldwide Media Service Inc., 30 Montgomery St., Jersey City , NJ 07302; 201-332-7100.

Estimated print run 160,000.

Miller Freeman
A United News & Media publication

ABP American Buisness Press

Printed in the USA

BPA International

Is It Real Time, or Is It Microsoft?

More than once, Microsoft's marketing knuckles have been rapped for its nasty little habit of preannouncing products. Not that rebukes seem to matter, if Microsoft's recent preannouncement for adding "hard" real-time capabilities to Windows CE is any indication. According to a somewhat nebulous press release, Windows CE will be a "hard" real-time operating system with the release of Version 3.0 sometime in the second quarter of 1999. Interestingly, at last fall's Embedded Systems West Conference, Microsoft was careful not to call Windows CE 2.0 "hard" real time at all. Then suddenly, like pigs sprouting wings, Microsoft was referring to WinCE as a hard RTOS at this spring's Windows CE Developers Conference. (Soft real time is more forgiving than hard. Soft real time can miss deadlines in cases where not completing tasks is more acceptable than a failure. Hard real-time deadlines, on the other hand, must always be met. At minimum, hard real time must be deterministic, have low latency, and support nested INTs.)

When asked about this in a *DDJ* web site "Online OP-ED" interview (http://www.ddj.com/), a WinCE product manager cleared things up, explaining that the 1999 release of WinCE 3.0 will be "true" hard real time, implying that WinCE 2.0 is some other kind of hard real time. Yes, by the most minimal of definitions, WinCE 2.0 is a RTOS—but it's about as hard as butter on your morning biscuits.

WinCE has a long way to go before it can truly be called hard real time—especially when compared to tried and tested RTOSs such as QNX, VRTX, VxWorks, pSOS, and the like. (For instance, some WinCE latency figures are measured at from 93–275 microseconds; under QNX, comparable figures are at about two microseconds.) In all likelihood, a total rewrite of the WinCE kernel will be required to bring WinCE up to par with real RTOSs. But for all we know, of course, that rewrite is underway.

If history has taught us anything about Microsoft, it is that the company has a hard time meeting promises when it comes to shipping operating-system products—especially when those products are announced more than a year in advance. Having more resources than most of us can imagine didn't necessarily get Windows 95/98/NT 5.0 out the door when promised, making you wonder why Windows CE should be any different.

So why does Microsoft keep on preannouncing operating-system products so far in advance? More than likely to freeze the marketplace until a minimal implementation of what's promised can be delivered. Clearly, that marketplace would be better served by walking the walk, instead of talking the talk.

***

About the same time Microsoft was exhibiting chutzpah in the real-time realm, O'Reilly & Associates was down the road patting itself on the back over its self-proclaimed "historic" Open Source Summit. According to its press releases, O'Reilly brought together "heavyweights of the Internet software community…to explore ways of expanding the use and acceptance of open source software development."

No question, this is an admirable goal that deserves all of our support. The invitation-only event included the likes of Linus Torvalds, Larry Wall, Brian Behlendorf, John Ousterhout, Guido van Rossum, Phil Zimmermann, John Gilmore, Eric Raymond, Tom Paquin, Jamie Zawinski, Sameer Parekh, Eric Allman, Greg Olson, and Paul Vixie—each of whom deserves accolades for his contribution to the world of software development.

More noticeable, however, was who wasn't invited. If any single person deserves credit for launching the open source software movement, it's Richard Stallman of GNU and free software fame. An "open source summit" without Stallman is like a cheeseburger without the cheddar.

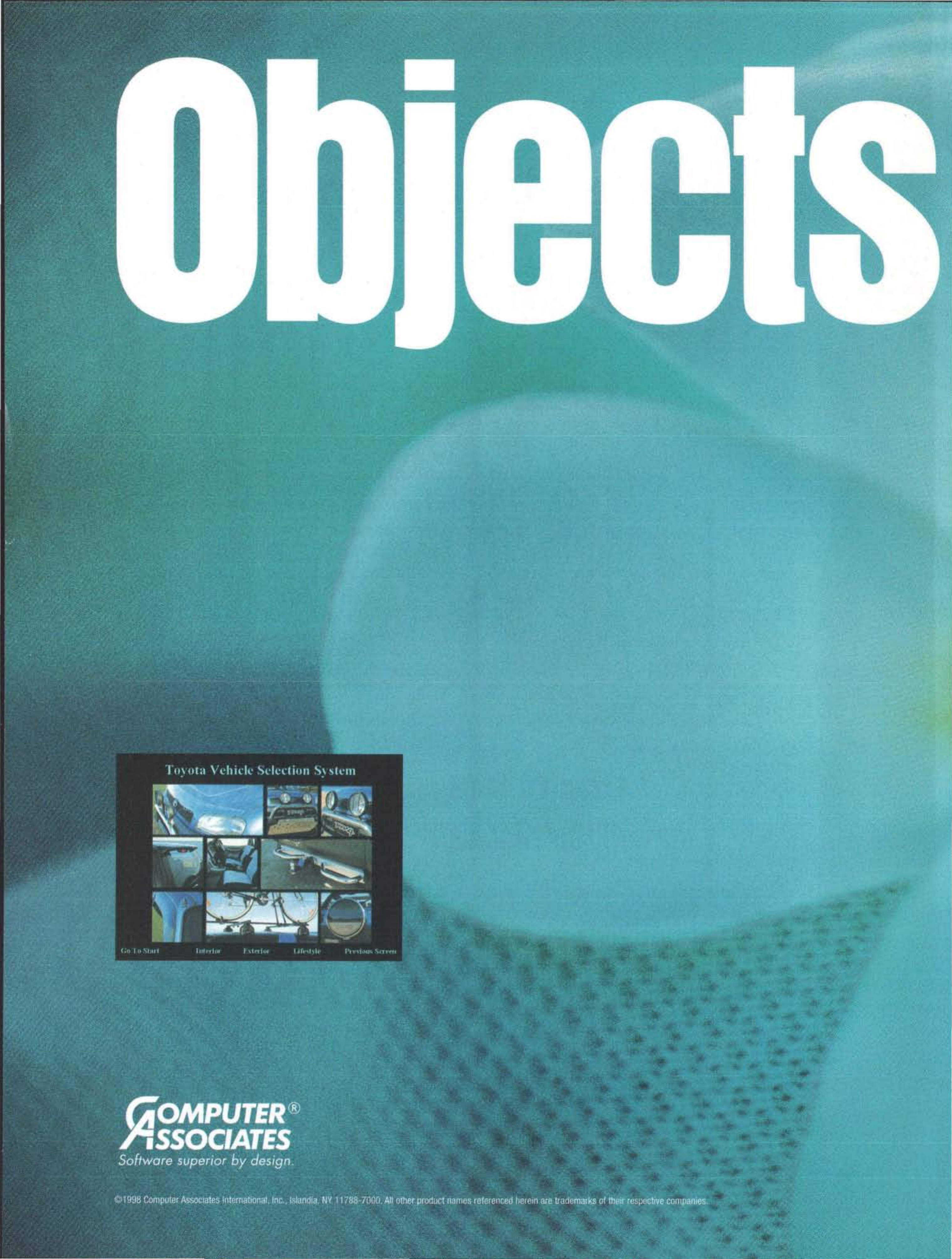
When, in response to a flurry of O'Reilly e-mail press releases, I asked by reply why Stallman wasn't invited, the net went suddenly quiet. Inquiring minds want to know.

***

A recent study by Software Success (http://www.softwaresuccess.com/) revealed a couple of interesting twists. The analysis, compiled by Software Success using data supplied by Dun & Bradstreet, showed that the total number of companies competing in the software industry grew from 58,779 in July 1997 to 68,765 in March 1998. For the first time since 1993 (when Software Success started tracking this data), the rate of growth of mid-sized companies was faster than that of startups. For instance, the number of companies with annual sales of under $500,000 (50,482) increased 12 percent since July 1997, the number with sales of $1 million–$5 million increased 42 percent, and those with $10M and up increased 83 percent. Software Success also found that the number of companies in the software-related services sector grew 45 percent to 19,542, reflecting a migration of some formerly product-based companies to the services sector.

Jonathan Erickson

Jonathan Erickson
editor-in-chief
jerickson@ddj.com



Objects
Toyota Vehicle Selection System
Go To Start
Interior
Exterior
Lifestyle
Previous Screen
COMPUTER®
ASSOCIATES
Software superior by design.

r Here.

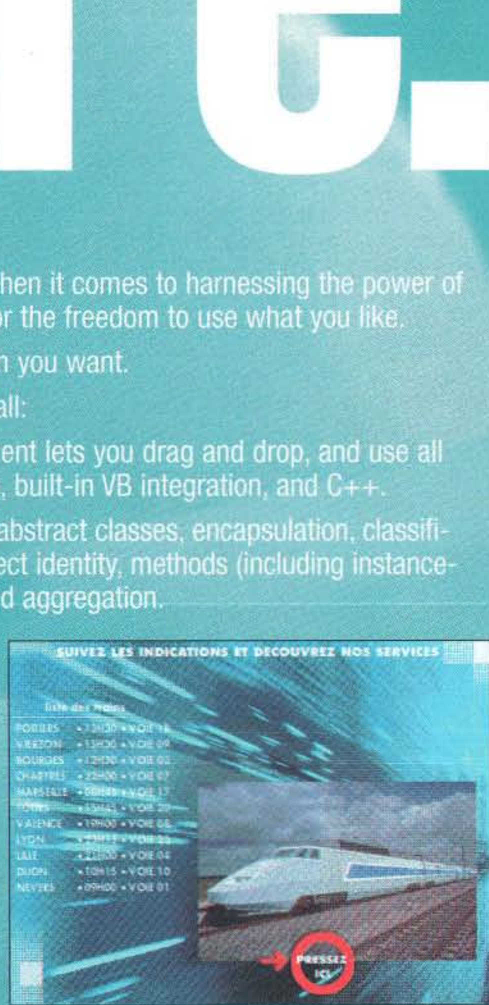
Everyone agrees: the future belongs to objects.

But today's application developers have limited choices when it comes to harnessing the power of object technology without sacrificing performance, flexibility, or the freedom to use what you like.

Jasmine™ gives you the power you need with the freedom you want.

It's the first complete and pure object solution that has it all:

- The industry's easiest integrated development environment lets you drag and drop, and use all your favorite tools: native Java support, ActiveX controls, built-in VB integration, and C++.
- Jasmine is a powerful, true object database. It supports abstract classes, encapsulation, classification, inheritance (both single and multiple), unique object identity, methods (including instance-level, class-level, and collection-level), polymorphism, and aggregation.
- An industrial-strength architecture featuring SMP support, backup and restore, security, and transaction management.
- Built-in multimedia and Internet support: not only can you build and package the next generation of multimedia business applications; you can run them everywhere: Internet, intranet, extranet, client/server —all without recompiling.



Jasmine is real. A proven, complete object-oriented database and development environment. Unlike hybrid or partial object solutions, Jasmine actually works.

So you can shorten your time to market and gain a distinct competitive advantage.

Try it out—pick up the phone right now. Because Jasmine is ready today.

Are you?

Call 1-888-7 JASMINE for your FREE Developer Edition CD or visit www.cai.com

Introducing Jasmine™

Objects @ Work™

Y2K Challenges

Dear *DDJ*,

In the article "Date Compression and Year 2000 Challenges," by Robert L. Moore and D. Gregory Foley (*DDJ*, May 1998), I was surprised to see the Windowing concept referred to as a preferred method for meeting the much closer Year 2000 deadlines. Surprising, only in that I've previously suggested the same method, and your discussion of the approach shows I'm not totally out of touch with reality.

In Robert and Gregory's discussion, one aspect of Windowing may benefit from an alternate approach.

In dealing with sorting of Y2K-deficient data, they mentioned an interpretive program would need to be activated for converting non-Y2K data into a sortable Y2K-compliant format. A secondary interpretive program would then need to be written for converting the sorted Y2K-compliant output back into its native format. Instead of this approach, if achievable, the following would be preferred.

Have the OS *sort* utility modified to include a sort option that incorporates a Windowing interpretation. In the JCL, a one-byte field would indicate that sort requires the sort Windowing option. This field (code) would be available for each sort field. As with Windowing, the sort parameters would contain a singular field for specifying the Windowing (pivot) year. The sort would perform an on-the-fly Y2K interpretation while sorting the data, without actually modifying or expanding file contents.

I realize many OSes may be in use and are no longer upgradable. With the billions of dollars which will need to be spent on Y2K compliance, I would think enough clout could be established to coerce someone to make the required modifications.

An alternative, the interpretative programs (used before and after each sort) described in the article could be designed to utilize the JCL passed codes, similar to the sort parameters context in order to determine which fields need to be converted. By using variable parameters of this nature, it wouldn't be necessary to write a separate set of programs for each sort.

Storing the pivot year in the Working Storage Section was also recommended. If only one program needed modification for Y2K compliance, that would be great. However, thousands of programs are going to be modified. To adhere to a Sliding Windowing concept, storing the pivot year in Working Storage would require massive modifications just to affect a new Window. Perhaps a singular file containing the definition of the Windowing method/period could be created (for storage of any periodic variable data). Then each affected program would open the file, extract the windowing data, then store the data in Working Storage.

After this was integrated into all programs utilizing the Sliding Windowing concept, modifying the pivotal year file would simultaneously affect all programs.

Wayne H. Wilhelm
whw96sv@cardnet.stark.k12.oh.us

Dear *DDJ*,

One problem with all the compression schemes mentioned by Robert L. Moore and D. Gregory Foley in their article "Date Compression and Year 2000 Challenges" (*DDJ*, May 1998) is that human readability is lost. ASCII only makes use of the seven least significant bits of each word. Using the most significant bit from each of the six characters used to represent a date by the MMDDYY method, and using a base year of 1900, we can extend the present method to 64 centuries. Hopefully in that time we can work out a better system. Dates printed by a routine that strips off the most significant bit will still be human readable.

Lloyd C. Brown
Lloyd.Brown@gat.com

Dear *DDJ*,

I congratulate Robert L. Moore and D. Gregory Foley on their clear, well-written article "Date Compression and Year 2000 Challenges" (*DDJ*, May 1998) that focuses on the fundamental engineering problem of the Y2K "situation." At work, I have had to complete many spurious Y2K forms and questionnaires from customers who just don't get it, and who have latched on to the four-digit year as a mantra to protect themselves from Y2K ruin. With Robert and Gregory's article, perhaps I can teach them to converse rationally about the subject (one can always hope).

However, I was disappointed about a slight omission in the discussion—the issue of backward compatibility of storage. As mentioned in the article, there are two goals in programming a Y2K fix: to provide a representation for all dates the system could possibly need, and to do this with a minimum of programming effort (including software maintenance). The authors also mention that compression methods can reduce the amount of coding required to fix Y2K problems. You can reduce that effort even further if you don't have to convert all your persistent data to a new representation.

All of the compression methods provided in the article use the entire "value space" of each representation. As such, they all collide with the legacy representation. For example, the six characters "012001" could mean January 20, 1901 (MMDDYY) or January 1, 1912 (CYYDDD) or January 20, 12337 (MMDD 16b-year). So there is no way to examine a date to determine the encoding scheme used. Implementing these representations requires that all existing data be converted before the new software may be used, and that the old software is fully retired before the change.

Namespace techniques can be used to remove this burden, by designing the new representation to be complementary with the legacy representation. As mentioned in the article, the MMDDYY format makes very sparse use of the 48 bits required for storage; all of the methods described by the authors can be modified to exclude the normal MMDDYY representations from their "value space" and still retain sufficiently large ranges of dates. For instance, modify the CYYDDD format so that values of C start at "2"; values of "0" and "1" would indicate data in the old format. This still provides 900 years of dates, but allows the program to read data in both CYYDDD and MMDDYY formats. Similar tricks will work with each of the other formats described—I leave the details as an exercise to the reader.

By using a backward-compatible compression scheme, the need for updating existing data sources to the new representation is removed. To implement the fix, we only need to reprogram the data interface (read from/write to storage), possibly adjusting the internal date format and user output to account for the increased range of dates. Then release the revised program to users. In my experience, this is the minimum effort required to correct a Y2K deficiency.

Curtis S. Carney
awiggin@slip.net

Java and CORBA

Dear *DDJ*,

In "Building Distributed Applications with Java and CORBA" (*DDJ*, April 1998), Bryan Morgan does a good job with outlining to intricacies of CORBA. I do find, however, that I cannot agree with some of his statements and findings.

First, CORBA is not a vendor-independent operating system. The Object Management Group (OMG) never intended CORBA to re-

(continued from page 10)
place the core operating system of any node in an n-tier client/server environment. Bryan's statement can leave someone thinking that CORBA can replace NT or UNIX. Had Bryan stated that CORBA provides a vendor-independent environment for inter-object management across a network, I would have agreed with him.

Secondly, the proper use of the Internet Inter-ORB Protocol (IIOP) is somewhat of a religious war among CORBA proponents. Bryan flippantly discusses IIOP as a "wire-level protocol that resides on top of TCP/IP." He further states that IIOP "lets one vendor's CORBA 2.0-compliant ORB exchange objects with another's." Bryan is certainly stating the promise of IIOP rather than the fact. Let's look at the facts:

- Since CORBA is a suite of guidelines and does not dictate how a vendor should implement its CORBA-compliant solutions, inter-operability among the various ORBs is less than ideal — even with IIOP.
- IIOP is a good beginning toward addressing ORB inter-operability, but falls short of synchronizing such CORBA services as security and time across ORBs. Companies building CORBA-based applications are advised to choose a single ORB vendor and remain as homogeneous as possible. Mixing and matching ORBs is risky business in today's client/server world.
- One of the original intentions of IIOP was to build a bridge between CORBA and the Distributed Computing Environment (DCE). CORBA proponents recognize that while DCE is falling from favor in the n-tier client/server environment, it provides mature network services for building and managing client/server applications. DCE, unlike CORBA, is an industry standard rather than a suite of guidelines. As an industry standard, DCE limits the implementation variation across vendors. Moreover, some of the most robust security and authorization facilities have grown out of the DCE standard, like Kerberos.
- Many CORBA-compliant vendors currently have stable products for CORBA 1.0 — fewer have stable products for CORBA 2.0. Since IIOP is part of the CORBA 2.0 specification, it would not make sound business sense to use IIOP to integrate ORBs from two or more vendors without knowing how each vendor has implemented their respective CORBA services (this defeats the purpose of encapsulation at the ORB service level, a foundation of object management).

CORBA is certainly the wave of the future. Since CORBA is an evolving specification, it is important that forethought and prudence are used to ensure that we build feature-rich and robust object-based client/server applications. As Java replaces C++ as the developer's tool of choice for building client/server applications, we must create greater awareness of what is real and doable, versus what is promised.

Richard S. Kravchuk
richard.kravchuk@ey.com

Window Sizes and the Registry

Dear *DDJ*,

Thanks to Al Stevens for the info in his April 1998 "C Programming" column on how to solve the problem with a window that could either be maximized or minimized. I had a clean install of Microsoft office on my machine. The only problem was that Microsoft Photo Editor refused to be anything but maximized or minimized. After reading Al's column, I searched the registry and found InitialPosition=65500,2,66112,565. I deleted that and all is well now. Not a big deal, kind of annoying, so I never went too far in finding out the problem.

Kevin Peck
KPeck@bridge.com

DDJ

9.0

MORE THAN 50+ FILE FORMATS

JPEG
DICOM
DIB
EPS
AVI
ICO
CUR
DCX
IMG
IOCA
MODCA
FAX[raw]
WINFAX
CALS raster
GEM
PNG
PSD
MAC
Photo CD (PCD)
FlashPix (FPX)
Exif
TGA
RAS
PCT
CMP
BMP
OS2 BMP
PCX
AWD

WMF
WPG
AVI
WAV
MIDI
MSP
VDA
GIF
-Embedded Text
-Animated
-Interlaced
-Transparency
TIFF 6.0
-MPT
-JPEG
-Packbits
-Huffman
-RLE
-LZW
-CMYK
-CCITT
-CCITT G31D
-CCITT G32D
-CCITT G4
-Bitonal
-Grayscale
-RGB
-YCbCr
-CIELAB

IMAGE PROCESSING

TRANSFORMS

Including resize, resample (interpolated resize), rotate (.01 degree), flip, invert, reverse, crop, underlay, shear, transpose, fill, auto deskew and combine bitmap (with mathematical and Boolean operations).

FILTERS

Including sharpen, blur, brighten, darken, invert, hue and saturation, intensity, contrast, gamma correction, histogram equalize, edge detect, line detect, emboss, mosaic, posterize, median and noise filters, spatial filter (which can be predefined such as gradient, laplacian, sobel, prewitt, shift and difference, line segment, or they can be customized), and more.

DRAWING

Draw directly to a bitmap surface using any of the windows GDI functions (such as TextOut, BitBlt, Ellipse, and Rectangle).

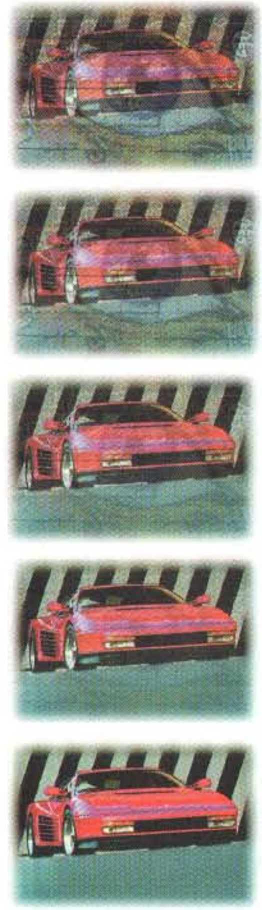
REGION OF INTEREST

Process only a specific portion of an image rather than the entire bitmap. Regions can be comprised of any combination of rectangles, ellipses, rounded-rectangles, freehand shapes, polygons, transparent color and more.

ADDING IMAGING?

not without

LEADTOOLS



If you have an imaging project in front of you, choose the #1 toolkit in the market. LEADTOOLS is an award winning imaging toolkit that puts more than 7 years of development and millions of lines of code at your finger tips. You will save countless hours and have access to the best imaging technology available. Imaging technology that Microsoft, Hewlett Packard, Corel, Xerox and thousands of others have chosen for their products. With LEADTOOLS, your project is almost done!

VIEWING & SPECIAL EFFECTS

LEADTOOLS provides optimized rendering of any image to all display devices with monitor calibrations, auto-dithering, scaling, zooming, scrolling, and animation. Choose from 113 Paint Effects, 64 Dissolve Effects, and 24 Transition Effects, including pushes, pulls, wipes, splits, blinds, crushes, rolls, circulars, diagonals, stretches and many more with delays, grain sizes, pattern brushes (up to 64 passes), a colored wand and a transparent color. The combinations are unlimited. Other Special Effects include: 3D Shapes, 3D Rotated Text, 3D Frames, Gradient and Pattern Filled shapes and much more.

INTERNET/INTRANET

LEADTOOLS features a Net Aware ActiveX and a Netscape plug-in for Internet/Intranet applications, including a Bitmap Datapath allowing images to be read from any URL, Progressive JPEG, Progressive CMP, and support for GIF interlace, transparency, animation, and embedded text. A FeedLoad function has been created to allow image data to be displayed as it is being transmitted across the net.

DATABASE

LEADTOOLS has specific features designed for the imaging database developer: VB data binding, 32-bit ODBC, a customized OLE 2.0 in-place server, Load/Save memory, and Load/Save file offset.

LEADTOOLS is backed up by a 30-day money back guarntee (US & Canada only) and FREE technical support is available via phone, fax, Internet, CompuServe or BBS.

SCANNING

LEADTOOLS supports high speed scanning using ISIS and TWAIN. TWAIN includes both 16 & 32 bit native and buffered RAM transfer modes.

PRINTING

LEADTOOLS performs all image processing necessary to print directly to any Windows supported printer, with the ability to print text and multiple images on the same page.

COMPRESSION

LEADTOOLS offers more compression options than any other toolkit on the market, in both standard and proprietary formats.

THE POWER OF LEADTOOLS

LEADTOOLS is a collection of more than 400 functions, properties and methods that provide low level functions for complete control, all the way to the highest level functions for ease of use. Imaging Common Dialog Boxes, (new to v. 9.0) greatly simplifies integration. Royalty free and available as 16 & 32 bit DLLs, 16 & 32 bit ActiveXs, and a VBX and includes extensive source code examples for Microsoft Visual C/C++, Borland C/C++, Microsoft Visual Basic, Borland Delphi, Microsoft Visual FoxPro, Microsoft Access, VB and Java Script.

NOW FEATURING EASY TO USE LEADTOOLS IMAGING COMMON DIALOGS

"LEADTOOLS is an indispensable tool, we used it in the development of our FrontPage application".

Tom Button
Director of Marketing,
Internet Platform & Tools division
Microsoft

"The file format support is phenomenal, LEADTOOLS gave Micrografx's applications (like CreataCard™ and Picture Publisher™) the flexibility to import and export a wide variety of established and new file formats with ease. Their format implementations are complete and performance Great!"

Andy Cohen
Director Software Development
Consumer Products
Micrografx

CUSTOMIZE NOW

FlashPix™ MODULE

Integrate the full FlashPix experience.
- FlashPix viewing transforms.
- FlashPix non-image data.
- FlashPix thumbnails.
- Load and Save portions of an image.

CUSTOMIZE NOW

VIDEO MODULE

Integrate extended multimedia capabilities.
- Motion video editing.
- Record, Create, Edit, Play and Save AVI files.
- Record, Play, and save WAV files.
- Play Midi files.

ALSO AVAILABLE

PRO EXPRESS

Top of the line LEADTOOLS toolkit.
- Document Imaging.
- Annotation.
- Xerox TextBridge® OCR.
- Bi-tonal Filters.

Order now: 800-637-1837

900 Baxter St. Charlotte, NC 28204
704-332-5532 Fax: 704-372-8161 CompuServe:"GO LEADTECH"

LEAD TECHNOLOGIES INCORPORATED

Special Offers, Evaluations, Demos & more! **http://www.leadtools.com/**

Paradise Pick of the Month!

ActiveReports
by Data Dynamics

For VB developers who need to generate a wide variety of reports from their applications, ActiveReports is an ActiveX Designer that is powerful and easy to learn. Unlike any other report writer, our product takes full advantage of VB's powerful, familiar language and data binding capabilities.

NEW!

$275

Paradise No. D03 0210-FT

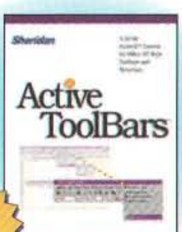
$100 Rebate*

$329.95*

Paradise No. M47 0112-FT

Visual Studio 97 Pro
by Microsoft Corp.

** Price after $100 manufacturer's rebate.*



New Product

$145

Paradise No. S16 0810-FT

ActiveToolBars
by Sheridan Software Systems

The Hottest Development Tools! Order Now!

Programmer's Paradise®

the developer's *definitive* source for software!

WISE Installation System Enterprise Edition
by WISE Solutions

WISE INSTALLATION SYSTEM *Enterprise* EDITION

The WISE Installation System Enterprise Edition incorporates the award-winning WISE Installation System, and the tools required for any type of application deployment. The Enterprise Edition includes SmartPatch, WebDeploy, and SetupCapture. The WISE Installation System creates professional installation programs for Windows, Windows 95, and Windows NT. WISE is a completely Windows-based installation editing/testing environment.

Paradise No. G10 0230-FT

$645

PluginWizard™
by Asgard

Harness the power of ActiveX controls in Netscape Navigator. Asgard PluginWizard will quickly generate Netscape Plugins from your own or 3rd party ActiveX controls. The standard edition directly creates a non-scripting plugin executable, while the professional edition generates full LiveConnect plugin source code. Royalty free!

NEW!

Paradise No. A66 0110-FT

$235

Visual SlickEdit
by MicroEdge

This award-winning editor increases development productivity, reduces costs of software maintenance and improves software quality through powerful features, software standardization and compatibility with your existing environment.

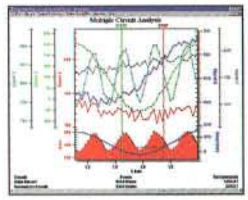
Software Standardization

With its multi-platform presence, integration with industry leading development environments and compatibility with version control systems, Visual SlickEdit provides your entire organization with a standard coding environment.

Paradise No. M39 0122-FT

$285

ProEssentials™
by GigaSoft®



16- and 32-bit DLL, OCX, VCL, and VBX interfaces providing charting functionality with consistent visual quality, real-world practicality, and overall professional appeal. Suited for engineering, financial, data-acquisition, and information system development. Be sure to find the best tool for your important development needs. Download a **demo** [500K] or get a **fully functional evaluation edition** [3Meg] at **www.pparadise.com/publishers/gigasoft.**

Paradise No. G04 0110-FT

$345

ClassMagic
by Object Dynamics

Assemble Windows applications, COM components or DLL in C++, using your current tools, libraries and code! Cut in half the code you write—create modular, thread-safe classes, connect objects, assemble new classes from interconnected objects, package them together. Includes the ClassMagic engine, library of reusable software parts, source samples, complete documentation and much more.

New!

Paradise No. 014 0110-FT

$479

VBTools 6
by BeCubed Software

Update your 16-bit applications today with any of the more than 60 custom 16-bit VBX controls! You'll save bunches of time with the new International control because it assigns internationalized text to all loaded controls! And there's a Flow Charting control, a 2D Slider, a Floating Text Extender and more, plus many other enhancements!

New Version!

Paradise No. B30 0310-FT

$95

Multi-Edit 8
by American Cybernetics

Multi-Edit 8 is the only 32-bit source-code editor a programmer needs. Fully customizable, it ships with extensive multi-language support including Web languages, IDE and VCS Integration and syntax highlighting. Run compilers and other tools in the background. Cut keystrokes in half with language-specific Smart Indenting, Template Editing, and Construct Matching.

Paradise No. A30 0110-FT

$149

Spread v2.5
by FarPoint Technologies

A complete spreadsheet control for most environments that support a VBX, OLE control, or DLL. Use as a spreadsheet to obtain variable lines of data or display tables of information. Data-aware—connect to databases with the Access Engine and ODBC. Includes over 250 properties and our improved Spread Designer. Formulas, sorting, and full-print support too.

Paradise No. F02 0110-FT

$225

Order online 24 hours a day, every day at www.pparadise.com

Visit ***www.pparadise.com*** *for the full story on these and more great products! Use our product search button & order online 24 hours a day, every day!*

The Fastest & Easiest Way to Create Help Systems

by Blue Sky® Software

RoboHELP 6.0
The Best Selling Help Authoring Tool*

RoboHELP is the fastest and easiest way to create online Help for any platform. RoboHELP turns Microsoft Word into a full-featured authoring tool capable of creating professional Windows Help, WebHelp, Windows 98 Help (Microsoft HTML Help), Netscape NetHelp, printed documentation, intranet/Internet Web sites, and Windows CE Help—all from a single source.

** Based on independent market survey.*

RoboHELP Office 6.0
The Complete Help Authoring Solution

RoboHELP Office provides a complete Help authoring solution with a suite of powerful tools for authoring nine professional Help formats, including the new Windows 98 Help format. Includes the award-winning RoboHELP, RoboHTML (at no extra cost–$499 value), plus over 16 powerful Help authoring utilities.

RoboHELP Office
Paradise No. B13 0240-FT
$659**

*** Price after $100 manufacturer's mail-in rebate.*

RoboHELP
Paradise No. B13 0140-FT
$469

LEADTOOLS ActiveX Pro 32

by LEAD Technologies, Inc.

LEADTOOLS offers technology in all major imaging categories for color, grayscale and bitonal imaging, with comprehensive functionality in each category. LEADTOOLS is an integrated development toolkit with more than 500 functions, properties and methods. Common Dialog boxes make LEADTOOLS easier to use and add-on modules are available for additional FlashPix, OCR, and Video support.

Paradise No. L05 0132-FT
$335

RoboHTML

by Blue Sky® Software

Easily Create Windows 98 Help

RoboHTML is the only product specifically designed to create the new Windows 98 HTML Help standard. RoboHTML provides a rich WYSIWYG editor with full drag and drop support, automated project management, and complete testing features. Unlike HTML editors, RoboHTML has built-in support for all Windows 98 specific features such as: dynamic Table of Contents; multi-level Indexes; and Related Topics. Also supports easy creation of Popups, Navigation Buttons, Splash Screens, Shortcuts, Import existing WinHelp projects and more.

Paradise No. B13 0320-FT
$469

Doc-To-Help

by Wextech Systems

Create HTML Help, Windows Help for 95, NT 3.51/4.0 and 3.0/3.1, HTML, and printed documentation from one file, at the touch of a button, with Doc-To-Help 3. Wextech, the Microsoft acknowledged expert in HTML Help authoring, incorporates our experience in Doc-To-Help 3, enabling you to create great HTML Help.

Paradise No. W08 0120-FT
$399

KEDIT for Windows 1.5

by Mansfield Software Group

KEDIT *for Windows*

Includes both 32-bit Windows 95/Windows NT and 16-bit Windows 3.1 modules. KEDIT is a powerful general purpose text editor with redefineable keys, undo/redo, selective line editing, regular expression support, enhanced syntax coloring, column oriented editing, file locking, a macro debugger, an IBM XEDIT-compatible command set, prefix area support, and more! The macro language is a subset of REXX. Also available in DOS and OS/2 text mode versions.

Paradise No. MAN KE31-FT
$135

Codewright Pro

by Premia

You'll work faster and easier when you use the right programmer's editor, Codewright. Now you'll browse code faster with Outline Symbols. It gathers information about your code in the background. You'll juggle changes with ease, with Difference Editing. It lets you selectively combine the changes from two revisions. The new Bookmarks Window lets you view bookmarks by name and by file. Now with synchronizing technology for Delphi and Visual C++ IDEs. With the help of the API Assistant, making complex function calls is as simple as filling in a form.

Paradise No. P36 0111-FT
$225

List Pro

by FarPoint Technologies

A set of customizable List Box and Combo Box controls. Create header groups and sub header groups and/or multiple columns to create a very unique list control. Merge cells in rows with same text to make display more user friendly. Bind to supported databases using Microsoft Jet Database Engine or Microsoft Access ODBC. Virtual memory manager allows connection to any size database. Supports single or multi-select lists, embedded icons and bitmaps in columns and rows, plus drag-and-drop capabilities.

Paradise No. F02 0211-FT
$185

Paradise Pick of the Month!

TestTrack

by Seapine Software

TestTrack is the fastest and most complete multi-user bug tracking solution for Windows 95/NT. Tracks bug and feature requests, customers, users, test configurations, and more. Advanced features include e-mail notifications, e-mail bug import, duplicate bug handling, release note generation, and much more. Distribute TestTrack's standalone bug reporter to your customers to automate customer support. With all of its power, TestTrack remains the easiest bug tracking solution to use and maintain.

TestTrack
Paradise No. S96 0210-FT
$469

A Personal Guarantee!

"If you're not happy, we're not happy! You have my personal guarantee that we'll provide you with outstanding products, support and service."

–Roger Paradis
President & CEO
Programmer's Paradise, Inc.

Here's just a small sampling of what you'll find in the Programmer's Paradise catalog. To find all the tools you need to succeed call us today!

ORDER NOW

or call for your FREE catalog!

1-800-445-7899

Call for shipping charges/return policy.
Prices subject to change without notice.

NASDAQ®: PROG

Dr. Dobb's News & Views

And the Winner Is...

The West Coast trounced the East Coast for the seventh time in ten tries at The Computer Museum's Tenth Annual Computer Bowl, a trivia contest held in Boston (West Coast fans gathered in Silicon Valley at Moffett Field's Hangar One to participate via video simulcast) that pits teams of five industry luminaries against each other. The West Coast, garbed in cowboy hats and led by Netscape cofounder Marc Andreesen, dominated the event, beating the frilly-shirt-wearing East Coast, 230–70. John Ratzenberger (aka Cliff Clavin, mailman and trivia buff from the TV show *Cheers*) was master of ceremonies of the event, asking questions ranging from the highly technical ("How many times more bandwidth does a T1 line have than a 56 kilobaud modem?") to the highly obscure ("How many microprocessors are there on Mars?"). Pride and fine food were at stake, which raises money for the museum—Sunnyvale, California, Mayor James Roberts won a lobster feast from a bet with Boston, Massachusetts, Mayor Thomas Menino. For more information, see http://www.computerbowl.org/.

Biometric Security Moves Forward

SAC Technologies (http://www.sacman.com/), a biometrics security company that provides technology for network and computer security without the use of pin numbers, passwords, or tokens, has received certification from the International Computer Security Association (http://www.icsa.net/). Certification was in the one-to-many Identification category. Identification is the process of comparing the biometric characteristics of an unknown individual against characteristics stored in a database to determine their identity. Identification asks, "Who is this?" and establishes whether more than one biometric record exists, thus denying an individual who is attempting to pass himself off with more than one identity.

Don't Blink

A "PIN-less" automatic teller machine (ATM) has gone online at the Nationwide Building Society bank in Great Britain. The system, designed by NCR, uses a biometric iris-identification system developed by Sensar (http://www.sensar.com/). To use the system, bank customers simply insert their ATM card into a reader and a camera mounted in the machine compares the customer's iris (one of the few human body parts to remain unchanged as aging occurs) with records in the databank. The process takes as little two seconds.

Sensar uses iris-recognition software developed by IriScan (http://www.iriscan.com/). The software is also being tested in Virginia by Spring Technologies as an automated fare-collection system in mass-transit applications. The goal of this automated system, called "TranScan," is to expedite commuter entry and exit at subway and train stations by minimizing and eventually eliminating the commuter's need to insert a card, pass, or token.

Macro Writing Contest

Premia Corp. has announced a macro writing contest for Premia's Codewright Programmer's Editor. The contest is being run in conjunction with the addition of Perl, AppBasic, and API (C-like) macros in Codewright 5.1. The grand prize for the best macro is $5000, or one of a number of other prizes. In addition, there will be first, second, and third place prizes for macros written in each of the three macro languages. Submissions must be received no later than August 1, 1998. Winners will be announced at the SD '98 East Conference in Washington, D.C. on August 18, 1998. For more information, see http://www.premia.com/.

E-Stamps on the Way

The U.S. Post Office has approved electronic postage stamps (e-stamps) for testing and, if things go as expected, we'll be printing our own stamps using PCs and the Internet. E-stamps include the postage amount, name and zip code of the local post office, date the postage was printed, and rate category (first class or whatever). In addition, e-stamps will have electronic bar coding of the same information as well as the identification number of the printing device and a digital pattern that will make each envelope unique and hard to counterfeit.

The system approved for testing, called "SmartStamp," was developed by E-Stamp Corp. (http://www.e-stamp.com/). Other approaches, such as PostagePlus from Neopost (http://www.neopost.com/), are coming online too. SmartStamp requires dongle-like hardware that fits into a printer port, serving as an electronic vault for postage. PostagePlus, on the other hand, requires no additional hardware. Customers will have an account with e-stamp companies and can download postage into this vault via the Internet.

Déjà Cygnus

Over the last few years, Metrowerks' CodeWarrior (http://www.metrowerks.com/) development tools have been extended from their Mac origins to include support for a wide array of languages (C, C++, Object Pascal, and Java), processors (including x86, PowerPC, MIPS, and Java VM), and systems (BeOS, PowerStation, Windows, and so on).

One of CodeWarrior's biggest competitors is GNU GCC, which has good support for cross-compilation to a variety of processors. To better appeal to companies that have standardized on GNU GCC, Metrowerks now officially supports the GNU GCC compiler from within the CodeWarrior environment (as an alternative to Metrowerks' own compiler). A new subsidiary, Quorum Technologies, has been formed for the express purpose of supporting GCC within CodeWarrior.

Cryptographers Crack Cell-Phone Code

Taking only about six hours of work, cryptographers at the University of California at Berkeley cracked Global System for Mobile Communications (GSM) codes, enabling them to "clone" a digital cellphone and make unauthorized calls from another phone. In the process, Ian Goldberg, David Wagner, and Marc Briceno also discovered indications that the code may have been intentionally weakened during its design. The GSM digital standard is the most widely used in the world, with more than 79 million phones in use.

Worldwide PC Sales Climb

According to a recent report by market-research firm Dataquest, sales of personal computers continue to grow at double-digit rates. Overall, says Dataquest, worldwide PC shipments were up 14.1 percent for the first quarter of 1998, compared with the same period in 1997. U.S. growth was 16.2 percent.

As for who's leading the vendor pack, Compaq maintained its market-share lead with 12.5 percent worldwide and 17.1 percent in the U.S. Dell Computer weighed in with an 11.7 percent in the U.S. Worldwide shipments by Hewlett-Packard and Dell were up 72 percent and 66.1 percent, respectively, over the last year.

Composing Reactive Animations

Programming for greater freedom of expression

Conal Elliott

There's no question that computer graphics — especially interactive graphics — is an incredibly expressive medium with potential beyond imagination. However, few people are able to create interactive graphics, so what might be a widely shared medium of communication is instead a tool for specialists. The problem is that authors still have to worry about how to get a computer to present content, rather than focus on the nature of the content itself. For instance, behaviors such as motion and growth are generally gradual, continuous phenomena; moreover, many such behaviors go on simultaneously. Computers cannot directly accommodate either of these basic properties, because they do their work in discrete steps rather than continuously, and they only do one thing at a time. Graphics programmers consequently have to bridge the gap between *what* an animation is and *how* to present it on a computer.

If the kind of programming in use today (like that described in the accompanying text box "Models versus Presentations" on page 25) is unsuitable for most potential authors, then we need to move toward a different form of programming. Alternative forms must give authors freedom of expression to say what an animation is, while invisibly handling details of discrete, sequential presentation. In other words, these forms must be declarative ("what to be"), rather than imperative ("how to do").

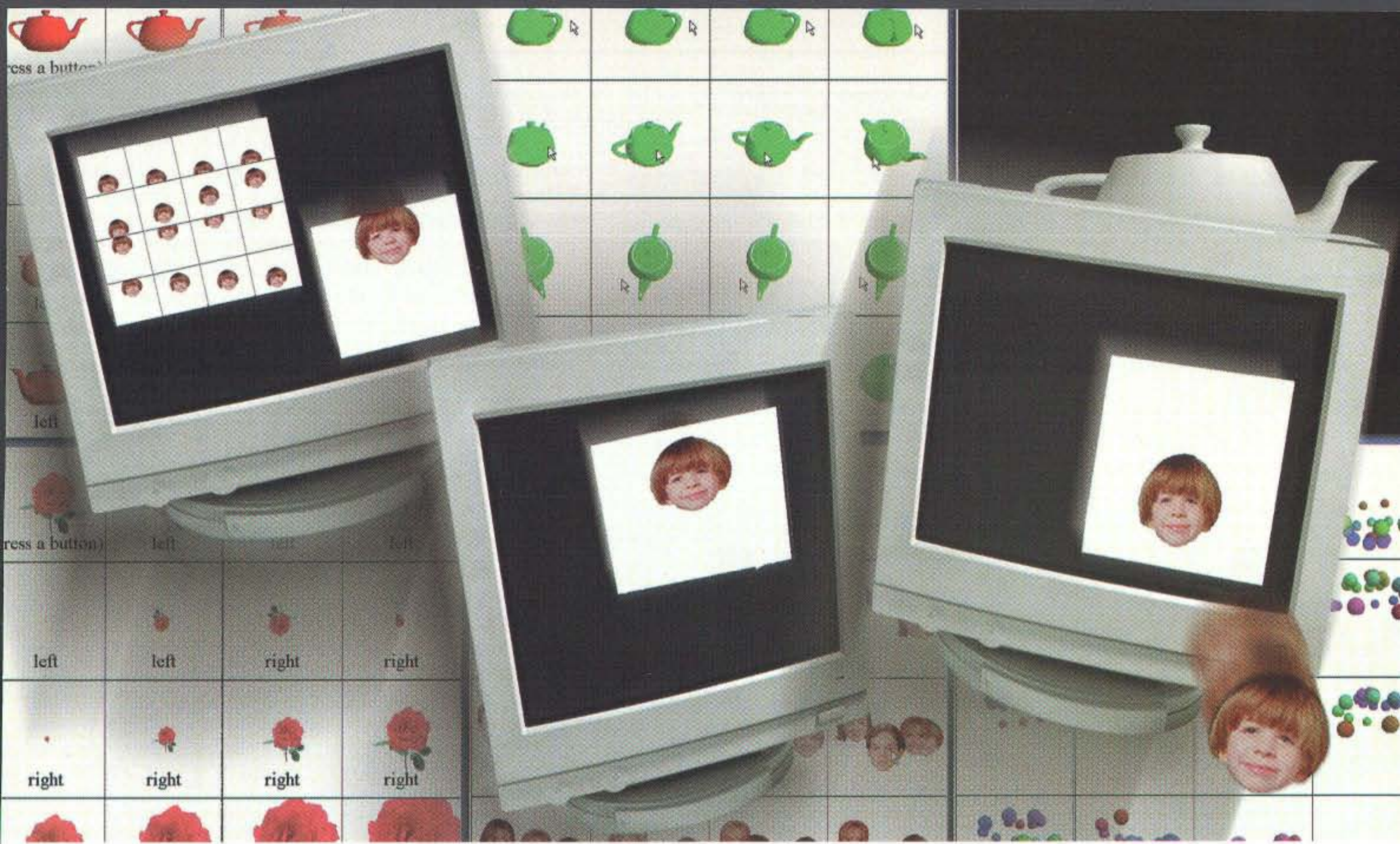
Conal is a member of the Microsoft Research Graphics Group. He can be contacted at conal@microsoft.com.

In this article, I present one such approach to declarative programming of interactive content. Fran (short for "functional reactive animation") is a high-level vocabulary that lets you describe the essential nature of an animated model, while omitting details of presentation. And because this vocabulary is embedded in a modern functional programming language (Haskell), the animation models are reusable and composable in powerful ways.

Fran is freely available (with source code) as part of the Hugs implementation of Haskell for Windows 95/NT (http://www.haskell.org/hugs/). Newer versions of Fran may be found at http://www.research.microsoft.com/~conal/Fran/. The underlying ideas form the basis of Microsoft's DirectAnimation, a COM-based programming interface accessible through conventional languages like Java, Visual Basic, JavaScript, VBScript, and C++. DirectAnimation is built into Internet Explorer 4.0, so you may already have it.

There are three ways you can experience this article:

- In this printed version, examples have an accompanying sequence of snapshots. By scanning them from left to right, top to bottom (first row, second row, and so on), you'll get a sense of motion.
- On the Web (http://www.research.microsoft.com/~conal/Fran/tutorial.htm), examples are illustrated by animated GIFs, showing animation over time, but not interactivity. That version of this article also contains additional discussion and several animations not in the printed version.
- Finally, you can run the examples and interact with or modify them. After installing Hugs (available at http://www.haskell.org/hugs/), double-click on the file tutorial.hs in the subdirectory lib\Fran\demos. At the > prompt, type "main" and press Enter. The examples will begin running. Press Spacebar, "n," or right arrow to advance to the next animation, and "p" or left arrow for the previous one. If you want to display just a single animation (*leftRightCharlotte*, for instance), then close the animation window and enter



"display leftRightCharlotte". You can alter the definition in an editor, save the result, enter ":r" to the Hugs prompt, and "$$" again to display the new version. For 2D examples having a user argument *u*, use *displayU* instead of *display*. Similarly, for 3D examples, use *displayG* if there is no user argument, and *displayGU* if there is a user argument.

The First Example

I'll start with the animation in Figure 1 called *leftRightCharlotte,* which moves Charlotte from side to side. Listing One (listings begin on page 20) defines a value called *leftRightCharlotte* to be the result of applying *moveXY* to three arguments. (In most other programming languages, you would instead say something like "moveXY(wiggle,0,charlotte)".)

The function *moveXY* takes *x* and *y* values and an image, and produces an image moved horizontally by *x* and vertically by *y*. All values may be animated. In this example, the *x* value is given by *wiggle*, a predefined smoothly animated number. *Wiggle* starts out at 0, increases to 1, decreases back past 0 to –1, and then increases to 0 again — all in the course of two seconds, and then it repeats, forever. The second line defines *charlotte* by importing a bitmap file, making it available for use on the first line as the second argument to *moveXY*.

Although this example isn't a masterpiece, it is nonetheless a complete animation program in just two short lines of code.

Similarly, Figure 2 and Listing Two define an animation of Patrick moving up and down. To get the vertical movement, I've used a nonzero value for the second argument to *moveXY*. Rather than using *wiggle*, you use *waggle*, which is defined to be just like *wiggle*, but delayed by half a second.

Figure 3 and Listing Three combine the two previous examples. The *over* operation glues two animations together, yielding a single animation, with the first one being over the second. Because I used *waggle* for *upDownPat* in this combined animation, Pat is at the center when Charlotte is at her extremes (and vice versa).

Composition

Composition is the principle of putting together simple things to make complex ones, then putting these together to make even more complex things, and so on. This building-block principle is crucial for making even moderately complicated constructions; without it, the complexity quickly becomes unmanageable.

Listings One through Three illustrate composition. I first built *leftRightCharlotte* out of *charlotte*, *wiggle*, and *moveXY*; then *upDownPat* out of *pat*, *moveXY*, and *waggle*. Finally, I built *charlottePatDance* out of *leftRightCharlotte* and *upDownPat*. A crucial point here is that when you make something out of building blocks, the result is a new building block in itself, and you can forget about how it was constructed.

There is a more powerful version of composition, based on defining functions. Listing Four, for instance, defines *hvDance* (for "horizontally and vertical dance"), which combines any two images, in the way that *charlottePatDance* combines *charlotte* and *pat*. Now you can give a new definition for the dancing couple that gives exactly the same animation: *charlottePatDance = hvDance charlotte pat*.

Having defined this generalized dance animation, you can go on to more exotic compositions. For example, you can take an animation produced by *hvDance*, shrink it, and put the result back into *hvDance* twice to make it dance with itself. As Figure 4 and Listing Five show, the result is pleasantly surprising. This example gives you a hint of how powerful it is to be able to define new animation functions. For instance, you could try *charlottePatDance*, stretched by a wiggly amount; see Listing Six(a). To prevent negative scaling, you take the absolute value of *wiggle*. Next, use *hvDance* again, but give it wiggly sized *charlotte* and *pat*. For visual balance, use *wiggle* and *waggle*; see Listing Six(b). Next, put Pat in orbit around a growing and shrinking Charlotte. To get a circular motion, use *moveXY*, with *wiggle* for *x* and *waggle* for *y*; see Listing Six(c).

As you may have surmised, *wiggle* and *waggle* are related to *sine* and *cosine* and defined as:

```
waggle = cos (pi * time)
wiggle = sin (pi * time)
```

The animated number *time* is a commonly used "seed" for animations and has the value t at time t. Thus, for instance, the value of *wiggle* at time t is equal to $sin(\pi t)$.

Rate-Based Animation

Up to now, the positions of animations have been specified directly. For instance, the definition of *leftRightCharlotte* says that Charlotte's horizontal position is *wiggle*.

In the physical universe, objects move as a consequence of forces. As Newton explained, force leads to acceleration, acceleration to velocity, and velocity to position. With computer animation, you have the freedom to ignore the laws of our universe. However, since animations are usually intended to be viewed by and interacted with by inhabitants of our own universe, they are often made to look and feel real by emulating Newtonian laws or simplifications and variations on them.

The key idea underlying Newton's laws and their variations is the notion of an instantaneous rate of change. Fran makes this notion available in animation programs. To illustrate rate-based animation, you can make Becky move from the left edge of the viewing window, toward the right, at a rate of one distance unit per second; see Figure 5 and Listing Seven.

The local definition of x here (introduced as a *where* clause), follows a style you'll see in the following definitions. To express an animated value that starts out with a value $x0$ and grows at

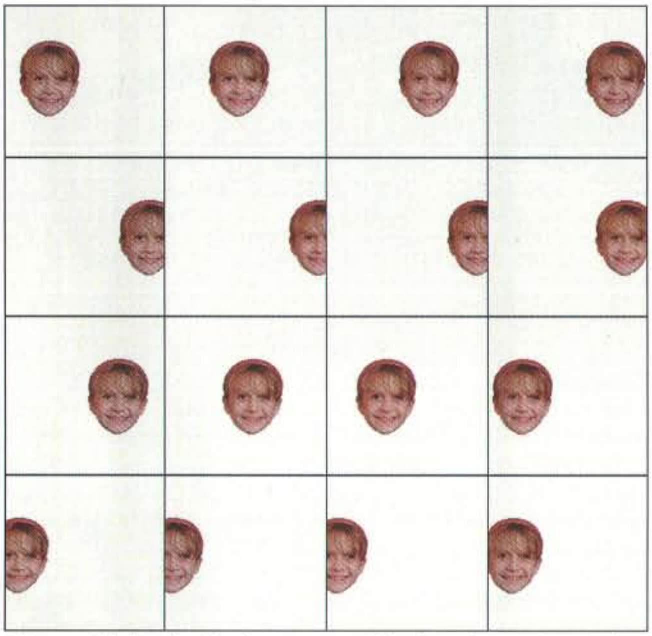


Figure 1: leftRightCharlotte *moves Charlotte from side to side.*

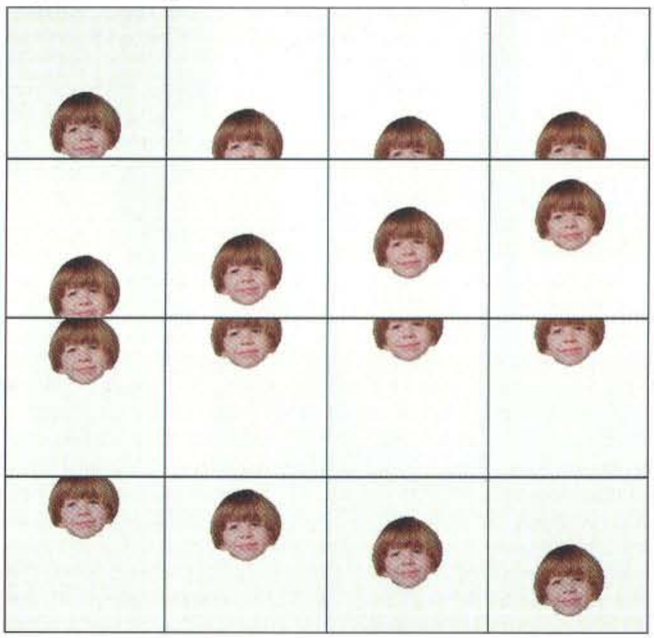


Figure 2: *Patrick moving up and down.*

Listing One

```
leftRightCharlotte = moveXY wiggle 0 charlotte
charlotte = importBitmap "../Media/charlotte.bmp"
```

Listing Two

```
upDownPat = moveXY 0 waggle pat
pat = importBitmap "../Media/pat.bmp"
```

Listing Three

```
charlottePatDance =
  leftRightCharlotte `over` upDownPat
```

Listing Four

```
hvDance im1 im2 =
    moveXY wiggle 0 im1  `over`
    moveXY 0 waggle im2
```

Listing Five

```
charlottePatDoubleDance = hvDance aSmall aSmall
  where
    aSmall = stretch 0.5 charlottePatDance
```

Listing Six

```
(a)
dance1 = stretch (abs wiggle) charlottePatDance

(b)
dance2 = hvDance (stretch wiggle charlotte)
                 (stretch waggle pat)

(c)
patOrbitsCharlotte =
   stretch wiggle charlotte  `over`
   moveXY wiggle waggle pat
```

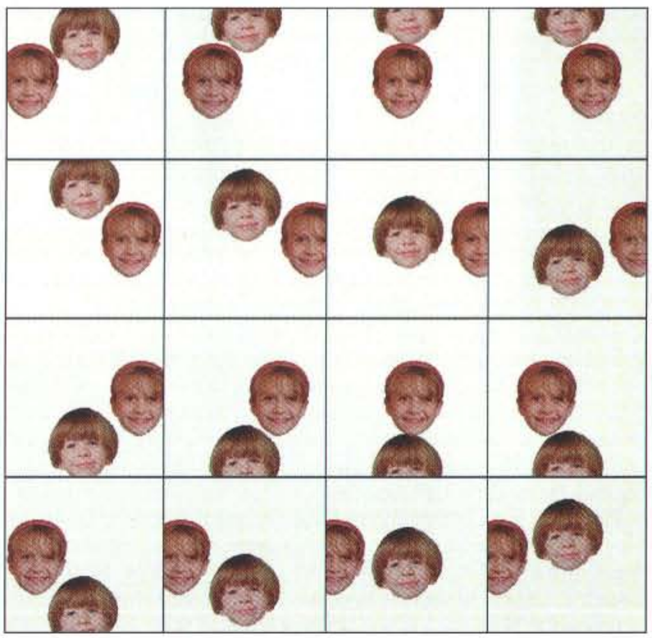


Figure 3: *Combining Charlotte and Patrick.*

(continued from page 20)

a rate of *r*, you say *x0 + atRate r u*. Here *u* is a "user", which is a Fran value that contains all user input and display update events. Rate-based animations require a user argument in order to give *atRate* a way of knowing when to start and how precisely to calculate value from rate. Unlike previous examples, this one can be displayed with *displayU*. To see this example, enter *displayU velBecky*.

In Listing Seven, Becky has a constant velocity, but with a little more effort you can give Becky a constant acceleration by providing a constant value for the rate of change of the velocity; see Listing Eight. In the definition of v, the "0 +" is unnecessary, but emphasizes that the initial velocity is zero.

The notion of "rate" is useful not just in one dimension, but in two and three dimensions as well. In Listing Nine, I control Becky's 2D velocity with the mouse. When you hold the mouse cursor at the center of the view window, Becky stays still. As you move away from the center, imagine an arrow from the window's center to the mouse cursor. Becky moves in that direction and her speed will be equal to the arrow's length. This kind of imaginary arrow is referred to as a "vector" and is the same type of quantity as a two- or three-dimensional offset, velocity, or acceleration. In 2D, a vector can be thought of as having horizontal and vertical (X and Y) components, or as having a magnitude (length) and direction. This time, I use *move*, a variant of *moveXY* that takes a 2D offset vector. (If a vector v is x units horizontally and y units vertically, then "move v im" is equivalent to "moveXY x y im.") The offset vector starts out as the zero vector, and grows at a rate equal to *mouseMotion*, which is the offset of the mouse cursor relative to the origin of 2D space (which you see in the center of the view window).

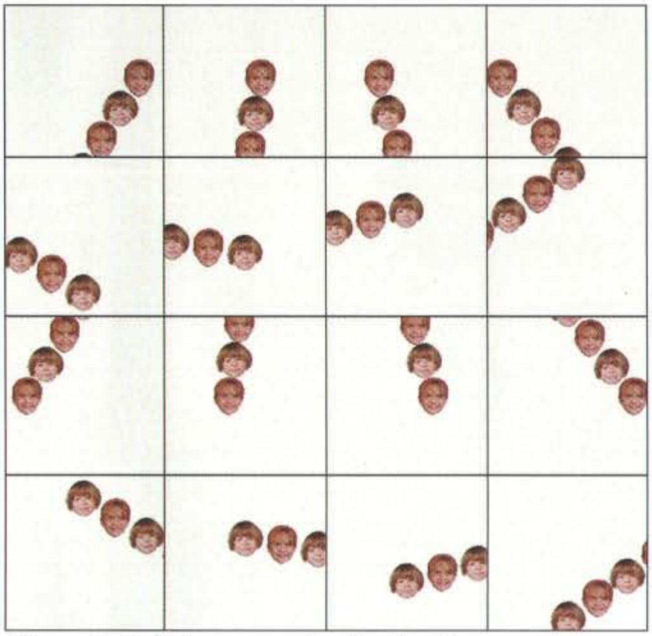


***Figure 4:** Defining new animation functions.*

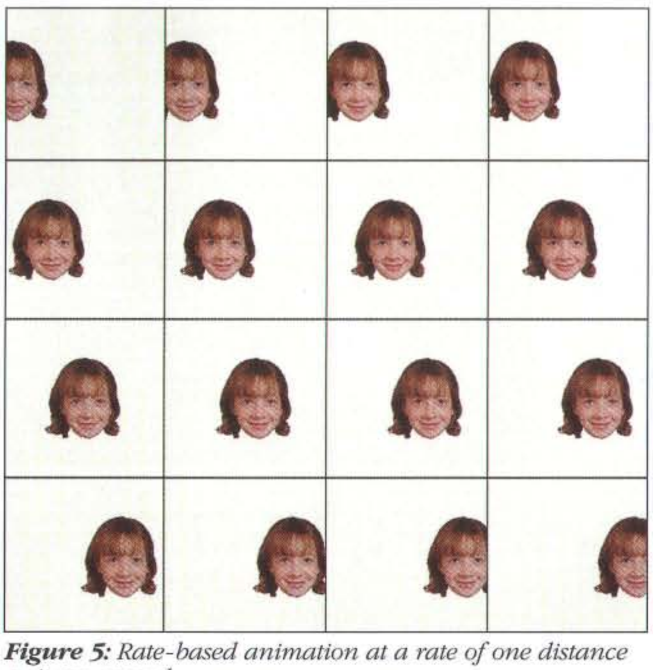


***Figure 5:** Rate-based animation at a rate of one distance unit per second.*

In the real world, the position of an object may affect its speed or acceleration. In Listing Ten, Becky is chasing the mouse cursor. The further away it is, the faster she moves. The only difference from Listing Nine is that the velocity is determined by where the mouse cursor is relative to Becky's own position, as indicated by the vector subtraction.

For fun, you can generalize the *beckyChaseMouse* function in the same way that *hvDance* generalized *charlottePatDance* earlier; see Listing Eleven. Then *chaseMouse becky* is equivalent to *beckyChaseMouse*, as you can verify by typing *displayU (chaseMouse becky)* at the Hugs prompt.

For more fun, try the same, but replace *becky* with some of the animations that appeared earlier (*leftRightCharlotte*, *charlottePatDance*, and *patOrbitsCharlotte*); see Figure 6 and Listing Twelve.

Next make a chasing animation that acts like it is attached to the mouse cursor by a spring. The definition is similar to *beckyChaseMouse*. In Listing Thirteen, however, the rate is itself changing at rate *accel* (acceleration). This acceleration is defined like the velocity was in the previous example, but this time, some

Listing Seven

```
velBecky u = moveXY x 0 becky
  where
    x = -1 + atRate 1 u
```

Listing Eight

```
accelBecky u = moveXY x 0 becky
   where
     x = -1 + atRate v u
     v =  0 + atRate 1 u
```

Listing Nine

```
mouseVelBecky u = move offset becky
   where
    offset = atRate vel u
    vel    = mouseMotion u
```

Listing Ten

```
beckyChaseMouse u = move offset becky
   where
    offset = atRate vel u
    vel    = mouseMotion u - offset
```

Listing Eleven

```
chaseMouse im u = move offset im
   where
    offset = atRate vel u
    vel    = mouseMotion u - offset
```

Listing Twelve

```
danceChase u =
   chaseMouse (stretch 0.5 charlottePatDance) u
```

Listing Thirteen

```
springDragBecky u = move offset becky
   where
    offset = atRate vel u
    vel    = atRate accel u
    accel  = (mouseMotion u - offset) - 0.5 *^ vel
```

(continued from page 22)

drag is also added. This tends to slow down Becky by adding some acceleration in the direction opposite to her movement. (Increasing or decreasing the "drag factor" of 0.5 in Listing Thirteen creates more or less drag.) The operator *^ multiplies a number by a vector, yielding a new vector that has the same direction as the given one but a scaled magnitude.

As usual, these declarative animation programs are straightforward because they say what the motion is, in high-level, continuous terms, without struggling to accommodate the discreteness of the computer used to present them. In contrast, imperative animation programs must explicitly simulate rate-based animation by making lots of discrete steps — accumulating approximations

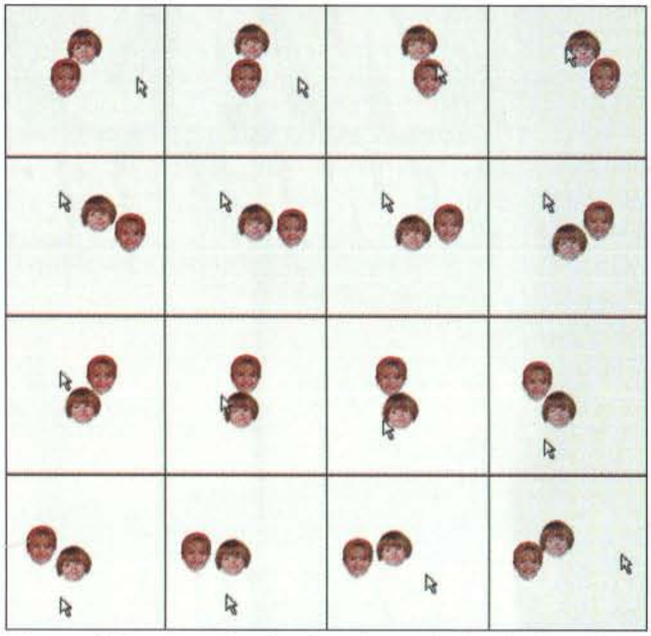


Figure 6: *Rate-based animation, but replacing Becky with animations such as* leftRightCharlotte, charlottePatDance, *and* patOrbitsCharlotte.

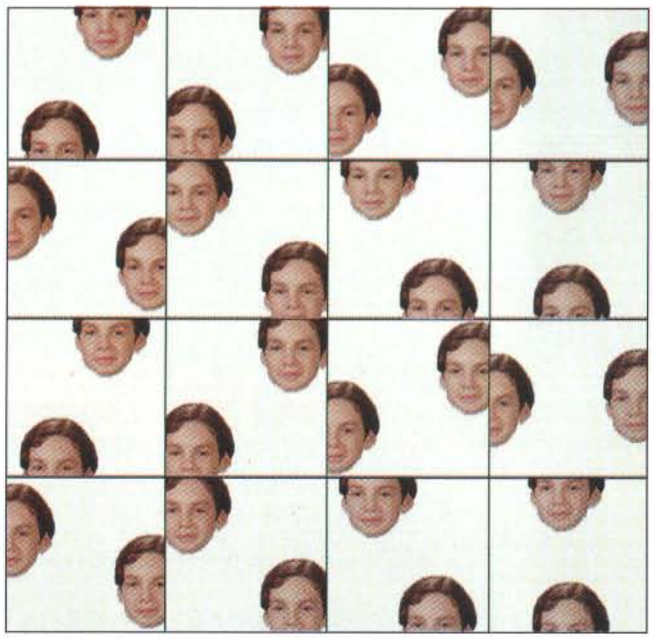


Figure 7: *Composition-in-time. Defining an orbiting animation, and then combining it with a version of itself delayed by one second.*

to the continuously varying forces, accelerations, and velocities—to approximate motion. Doing an accurate and efficient job of all this approximation work is a tricky task. With systems like Fran, you just describe the continuous motion in terms of continuously varying rates, and trust Fran to do a good job with the approximation. (Not good enough to fly an airplane or control dangerous machinery, but good enough for an effective illustration or game.)

Composition-in-Time

Operations such as *over* and *move* support the principle of composition-in-space. Composition-in-time is equally valuable. Figure 7 and Listing Fourteen, for instance, define an orbiting animation, and then combine it with a version of itself delayed by one second. Instead of delaying, you can speed it up; see Listing Fifteen. You can even delay or slow down animations involving user input. In Listing Sixteen, one Jake tracks the mouse cursor, while the other follows the same path, but delayed by one second.

Next you can build an animated sentence, following the mouse's motion path. As a preliminary step, use *delayAnims dt anims = overs (zipWith later [0, dt ..] anims)* to define a *delayAnims* function, which takes a time delay *dt* and a list *anims* of animations, and yields an animation. Each successive member of the given animation list is delayed by the given amount after the previous member. The definition of *delayAnims* introduces a few new Fran elements. The Fran *overs* function is like *over*, but applies to a list of animations rather than just two. Animations earlier in the list are placed over ones later in the list. The notation *[0, dt ...]* means the infinite list of numbers 0, *dt*, 2 *dt*, 3 *dt*, and so on. Finally, *zipWith* applies to a given two-argument function the successive values from two given lists. You use it here to delay the first animation in *anims* by 0 seconds, the second by *dt* seconds, the third by 2*dt* seconds, and so on. Finally, *overs* combines them into a single animation. Figure 8 and Listing Seventeen present a simple use of *delayAnims*. Next, use *delayAnims* (Listing Eighteen) to define *mouseTrailWords* that makes animated sentences.

The Haskell *words* function takes a string apart into a list of separate words. The Haskell *map* function takes a function (*moveWord*) and a list of values (the separated words) and makes a

Models versus Presentations

Here is a rough sketch of the steps you usually go through to program an animation:

```
Allocate and initialize window, various drawing surfaces and bitmaps
  repeat until quit:
        get time ( t)
        clear back buffer
        for each sprite (back to front):
        compute position, scale, etc. at t
        draw to back buffer
        fast copy ("blit") back buffer to front
  Flip back buffer to the screen
Deallocate bitmaps, drawing surfaces, window
```

These steps are usually carried out with lots of tedious, low-level code you have to write yourself. Most of this work is not about what the animation is, but how to present it. In contrast, Fran programs are only about what the animation is.

—C.E.

Listing Fourteen

```
orbitAndLater = orbit `over` later 1 orbit
  where
    orbit = moveXY wiggle waggle jake
```

Listing Fifteen

```
orbitAndFaster = orbit `over` faster 2 orbit
   where
    orbit = move wiggle waggle jake
```

Listing Sixteen

```
followMouseAndDelay u =
  follow `over` later 1 follow
   where
    follow = move (mouseMotion u) jake
```

Listing Seventeen

```
kids u =
   delayAnims 0.5
     (map (move (mouseMotion u))
          [jake, becky, charlotte, pat])
```

Listing Eighteen

```
trailWords motion str =
  delayAnims 1 (map moveWord (words str))
  where
     moveWord word = move motion (
                       stretch 2 (
                         withColor blue (stringIm word) ))
```

Listing Nineteen

```
flows u = trailWords motion
             "Time flows like a river"
   where
     motion = 0.7 *^ vector2XY (cos time)
                                (sin (2 * time))
```

Listing Twenty

```
flows2 u = trailWords (mouseMotion u)
              "Time flows like a river"
```

Listing Twenty-One

```
redBlue u = buttonMonitor u `over`
              withColor c circle
   where
    c = red `untilB` lbp u -=> blue
```

Listing Twenty-Two

```
redBlueCycle u = buttonMonitor u `over`
                   withColor (cycle red blue u)
                              circle
   where
    cycle c1 c2 u =
     c1 `untilB` nextUser_ lbp u ==> cycle c2 c1
```

Listing Twenty-Three

```
tricycle u =
    buttonMonitor u `over`
    withColor (cycle3 green yellow red u) (
      stretch (wiggleRange 0.5 1)
        circle )
   where
    cycle3 c1 c2 c3 u =
     c1 `untilB` nextUser_ lbp u ==>
     cycle3 c2 c3 c1
```

Listing Twenty-Four

```
jumpFlower u = buttonMonitor u `over`
                 moveXY (bSign u) 0 flower
flower = stretch 0.4
            (importBitmap "../Media/rose medium.bmp")
bSign u = selectLeftRight 0 (-1) 1 u
```

Listing Twenty-Five

```
growFlower u = buttonMonitor u `over`
                 stretch (grow u) flower
grow u = size
  where
   size = 1 + atRate rate u
   rate = bSign u
```

Listing Twenty-Six

```
growFlowerExp u = buttonMonitor u `over`
                    stretch (grow' u) flower
grow' u = size
  where
   size = 1 + atRate rate u
   rate = bSign u * size
```

new list by applying the function to each member of the list. The Fran *stringIm* function makes a picture of a string. I define the function *moveWord* locally to be the result of making a picture of the given word, using the Fran *stringIm* function, and moving it to follow the mouse. *delayAnims* then causes each of these mouse-following word pictures to be delayed by different amounts. Figure 9 and Listing Nineteen is a use of *trailWords* following a specified path, while Listing Twenty follows the mouse.

Reactive Animation

The animations presented to this point can be called "nonreactive" since they always do the same thing. A "reactive" animation, on the other hand, involves discrete changes due to events. To illustrate, you can make a circle that starts off red and changes to blue when the left mouse button is pressed.

An informal reading of the last line of Listing Twenty-One (also see Figure 10) is that the color *c* is red until you press the left mouse button, then becomes blue. For a more literal reading, you must understand that there are really two new binary infix operators here—*untilB* and -=>—which can be used separately or together. Implied parentheses are around *lbp u* -=> *blue*. The -=> operator, which can be read as "handled by value," takes an event (*lbp u*) and a value (*blue*), and yields a new event. In this case, the new event happens when the left button is pressed, and has value *blue*. The *untilB* operator takes an animation of any type (the color-valued constant animation *red*), and an event (*lbp u* -=> *blue*), whose occurrence provides a new animation of the same type.

Cyclic Reactivity

To make Figure 10 more interesting, you can switch between red and blue every time the left button is pressed. As Listing Twenty-Two shows, you do this with the help of a *cycle* function that takes two colors (*c1* and *c2*) and gives an animated color that starts out as *c1*. When the button is pressed, it swaps *c1* and *c2* and repeats (using recursion).

Listing Twenty-Two uses the operator ==>, which is a variant of -=>. This operator (which can be read as "handled with function") takes an event and function *f*. It works like -=>, but gets event values by applying *f* to event values from the event given to it. In this case, *f* is the *cycle* function applied to just two arguments, leaving the third (a user) to be filled in automatically (using ==>). The *nextUser_* function turns *lbp* into an event

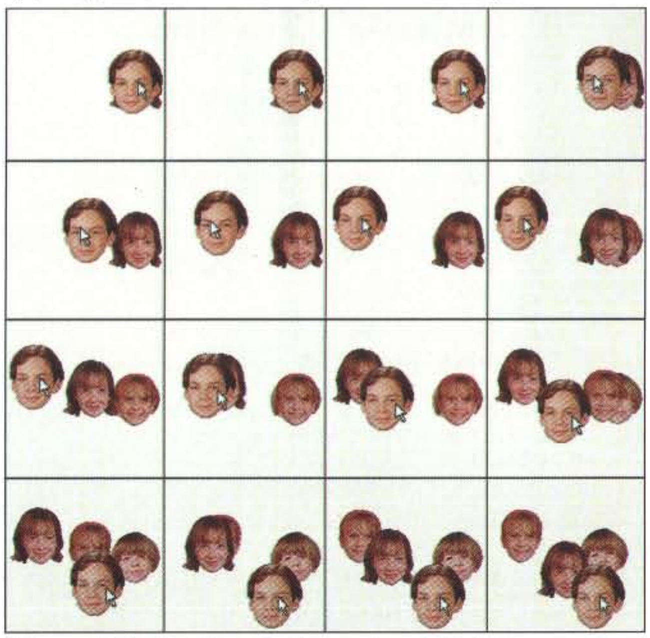


Figure 8: *Composition-in-time using* delayAnims.

(continued from page 26)
whose occurrence information is a new user, corresponding to the remainder of the user *u*. The color arguments get swapped each time "around the loop."

For variety, Listing Twenty-Three uses three colors, and changes the circle's size smoothly.

Selection

Figure 11 and Listing Twenty-Four present a flower that starts out in the center and moves to the left or right when the left or right mouse button is pressed, returning to the center when the button is released.

The function *bSign* is defined to be –1 when the left button is down, +1 when the right button is down, and 0 otherwise (thanks to *selectLeftRight*). You can use *bSign* to control the rate of growth of an image. In Figure 12 and Listing Twenty-Five, pressing the left (or right) button causes the image to shrink (or grow) until released. Put another way, the rate of growth is 0, –1, or 1, according to *bSign*. A simple change to the *grow* function (Listing Twenty-Six) causes the image to grow or shrink at a rate equal to its own size. *selectLeftRight*, used to define *bSign*, is also the key ingredient in defining *buttonMonitor* (Listing Twenty-Seven), which gives button feedback.

stringBIm turns an animated string into an image animation, which here gets enlarged, colored white, and moved down by a little less than half the window height.

selectLeftRight can itself be defined in terms of more basic functions, as in Listing Twenty-Eight. You use the conditional function *condB* to say that if the left button is down, use the left value, or if the right button is down, use the *none* value; otherwise use the *none* (*constantB*, which turns constants—nonanimations—into animations that never change).

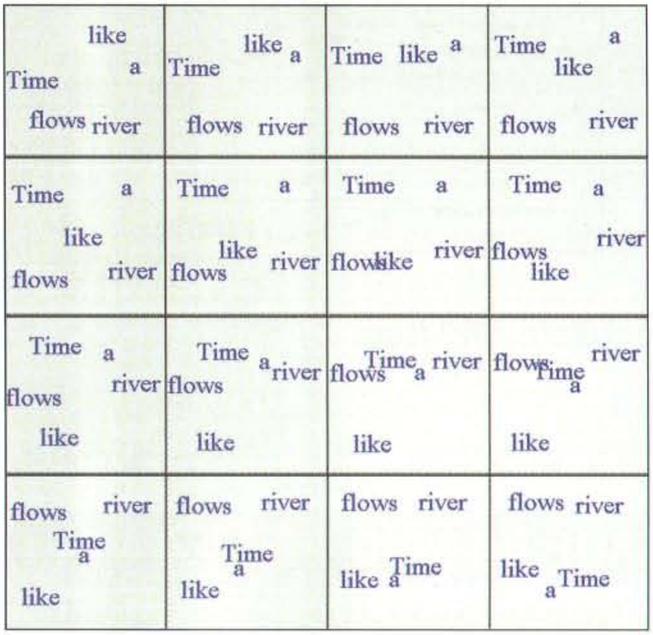


Figure 9: *Using* trailWords *following a specified path.*

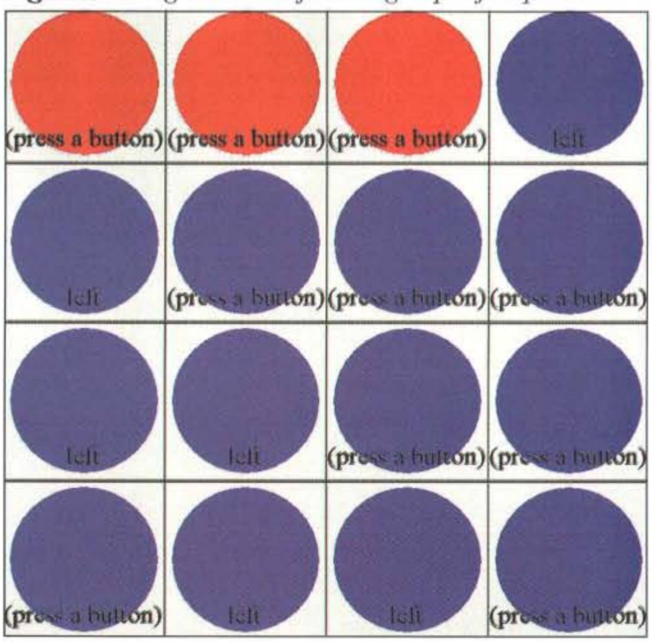


Figure 10: *The color* c *is red until you press the left mouse button, then it becomes blue.*

Listing Twenty-Seven

```
buttonMonitor u =
   moveXY 0 (- height / 2 + 0.25) (
    withColor textColor (
     stretch 2 (
     stringBIm (selectLeftRight "(press a button)" "left" "right" u)))
  where
    (width,height) = vector2XYCoords (viewSize u)
```

Listing Twenty-Eight

```
selectLeftRight none left right u =
   condB (leftButton u) (constantB left) (
     condB (rightButton u) (constantB right) (
       constantB none ))
```

Listing Twenty-Nine

```
teapot =
   stretch3 2 (importX "../Media/tpot2.x")
```

Listing Thirty

```
redSpinningPot =
  turn3 zVector3 time (
    withColorG red teapot)
```

Listing Thirty-One

```
mouseTurn g u =
   turn3 xVector3 y (
    turn3 zVector3 (-x) g)
  where
    (x,y) = vector2XYCoords (pi *^ mouseMotion u)
 mouseSpinningPot u =
   mouseTurn (withColorG green teapot) u
```

Listing Thirty-Two

```
spinPot potColor potAngle =
  turn3 zVector3 potAngle (
    withColorG potColor teapot)
```

Listing Thirty-Three

```
spin1 u = buttonMonitor u `over`
           renderGeometry (spinPot red (grow u))
                          defaultCamera
```

Listing Thirty-Four

```
withSpinner f u =
   buttonMonitor u `over`
   renderGeometry (f (grow u) u)
                  defaultCamera
```

Listing Thirty-Five

```
spin1 = withSpinner spinner1
  where
    spinner1 angle u = spinPot red angle
```

e-business

To get a **free demo CD**, and to find out more about Electronic Commerce visit us at www.software.ibm.com/eci

Are all your applications working together toward a common goal? Your CEO wants to compete on a whole new playing field: Electronic Commerce. The fastest way to get there is by connecting your existing applications to each other and to the Web. Enter MQSeries® from IBM. It allows you to get your ordering, shipping, inventory and accounting systems to work together as a back-end to your e-commerce Web servers, without writing a lot of connectivity code. MQSeries lets you pipe messages between applications, while it takes care of program interfaces and supports Java™, COBOL, C++, Lotus Notes® and SAP.® It's available on 25 platforms, including Windows NT®, HP-UX, Sun™ Solaris® and all IBM platforms. MQSeries helps guarantee message delivery, even when programs or networks fail. So if you're new to the e-commerce game or you've been there for a while, MQSeries can put your business on a new playing field. Find out about getting your apps working like a team at www.software.ibm.com/eci

IBM®

Solutions for a small planet™

(continued from page 28)

3D Animation

Declarative animation applies to 3D as well, and the 2D operations I've used to this point—*importBMP*, *moveXY*, and *stretch*—have 3D counterparts. As a first 3D example, *sphere = importX "./Media/sphere2.x"* defines a sphere in which the function *importX* brings in a 3D model in "X-file" format, as used by Microsoft's DirectX. It is just as easy to import a teapot; see Figure 13 and Listing Twenty-Nine. I used *stretch3* (a 3D counterpart to *stretch*) because the imported model was too small. Listing Thirty colors the teapot and makes it spin around the z- (vertical) axis.

Next, you can use the mouse to control the teapot's orientation. To do this, define *mouseTurn* to turn a given geometry *g* around the x-axis according the mouse's vertical movement, and around the z-axis according the mouse's horizontal movement, scaled by π Finally, as Figure 14 and Listing Thirty-One show, you apply *mouseTurn* to a green teapot.

You can also make teapots spin by controlling the rotation angle with the *grow* function, as in the growing flower examples. First, define *spinPot*, see Listing Thirty-Two, that takes (animated) color and angle and yields a colored, turning teapot. Then make a pot that spins one way when the left button is pressed, and the other way when the right button is pressed, using the *grow* function, and giving feedback with *buttonMonitor*, see Figure 15 and Listing Thirty-Three. *renderGeometry*, used here with a convenient default camera, turns a 3D animation into a 2D animation.

Additional spinning teapots will all have the general form of using the button monitor and rendering with the default camera. Rather than having to write several definitions, give the pattern a name. In Listing Thirty-Four, *withSpinner* takes a function as its first argument, and applies that function to the result

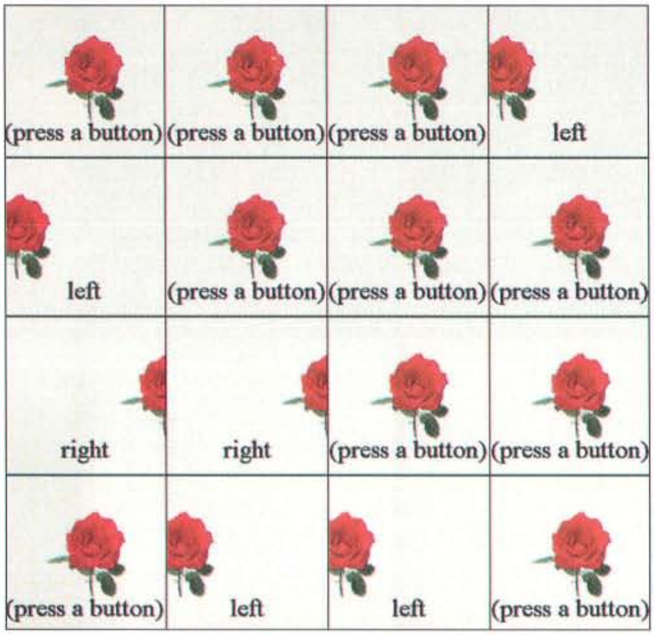


Figure 11: *Flower starts in the center and moves to the left or right when the left or right mouse button is pressed, returning to the center when the button is released.*

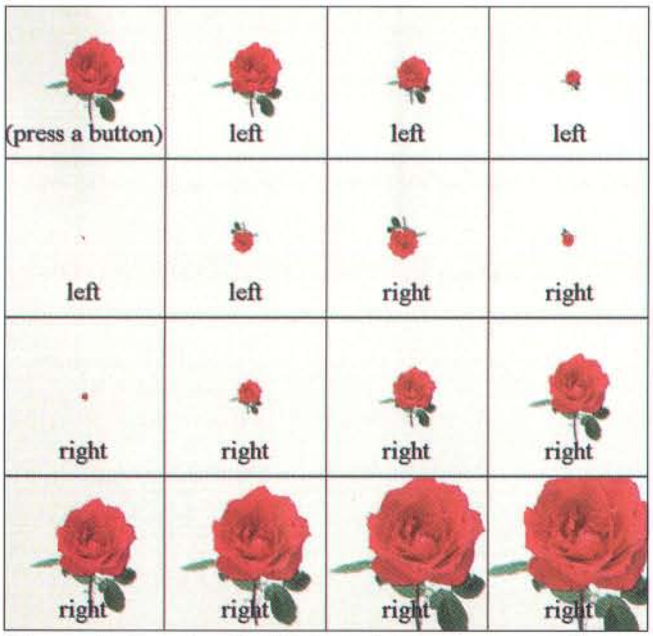


Figure 12: *Pressing the left (or right) button causes the image to shrink (or grow) until released.*

Listing Thirty-Six

```
spin2 = withSpinner spinner2
  where
    spinner2 potAngleSpeed u =
      spinPot (colorHSL time Ø.5 Ø.5)
              (atRate potAngleSpeed u)
```

Listing Thirty-Seven

```
sphereLowRes = importX "../Media/sphereØ.x"
movingLight =
   move3 motion (
    stretch3 Ø.1 (
     withColorG white (
      sphereLowRes `unionG` pointLightG)))
 where
  motion = vector3Spherical 1.5
             (pi*time) (2*pi*time)
potAndLight =
  withColorG green teapot `unionG` movingLight
```

Listing Thirty-Eight

```
delayAnims3 dt anims =
   unionGs (zipWith later [Ø, dt ..] anims)
```

Listing Thirty-Nine

```
potAndLights =
   slower 5 (
    withColorG green teapot `unionG`
    delayAnims3 (2/5) (replicate 5 movingLight) )
```

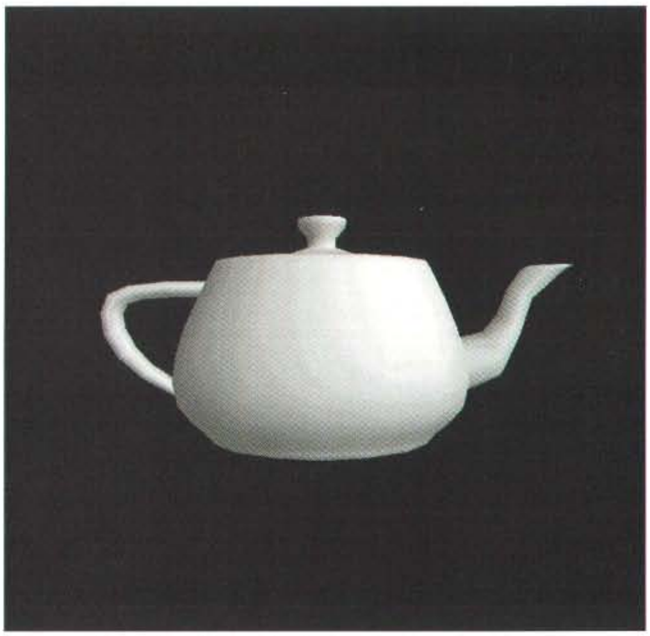
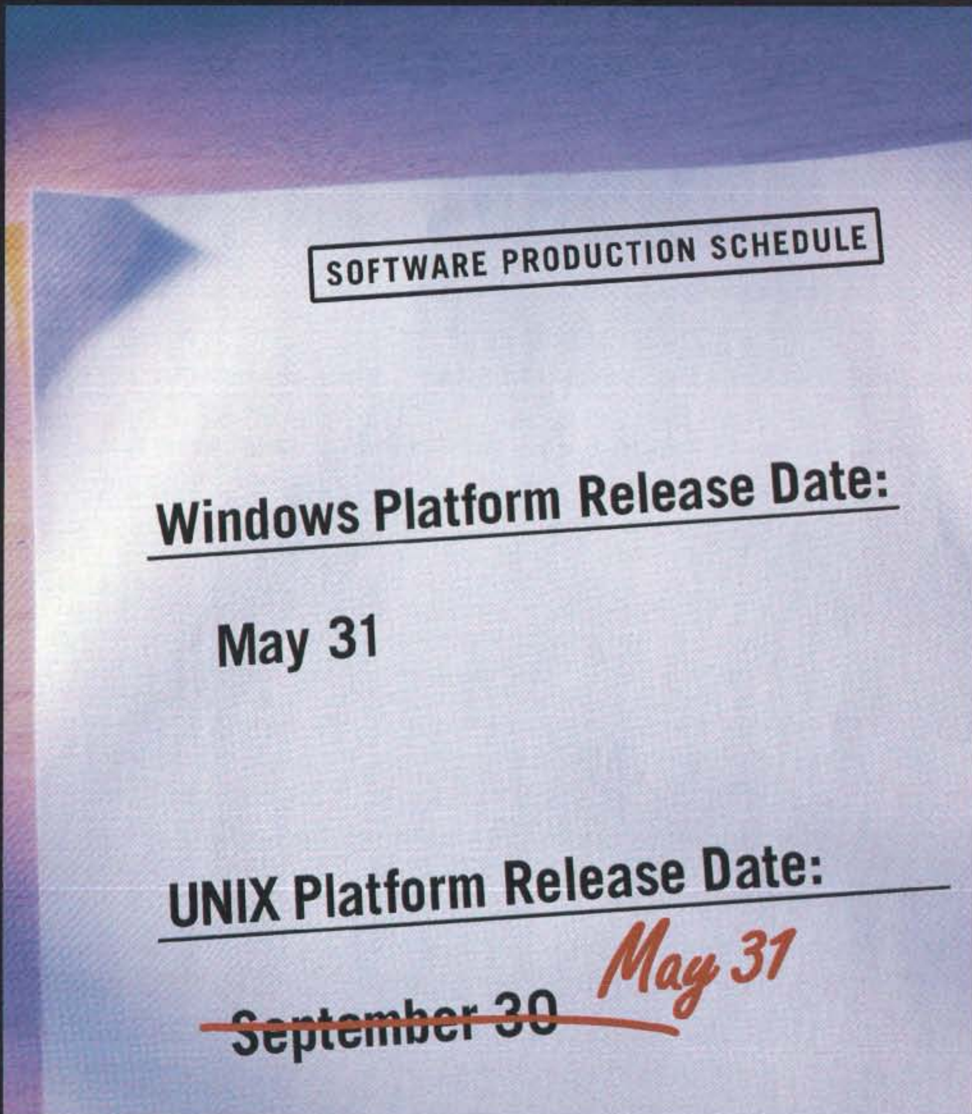


Figure 13: *Importing a teapot.*



USE MAINWIN TO PORT NT CODE TO UNIX, AND YOU'LL ONLY HAVE TO REWRITE ONE LINE.

Fast. Flawless. Flexible. When you use the MainWin cross-platform toolkit, you write apps for Windows once, then port the source code to UNIX, using UNIX-native WIN32 APIs. There's no need to double your programming efforts, no need to rewrite your code.

WWW.MAINSOFT.COM

Plus, you can still retain the same Test and QA staff, and even use the same documentation for both versions. The comprehensive MainWin environment ensures native performance to your existing UNIX customers, while speeding your own migration to the expanding world of Windows NT.

MAINSOFT

You will significantly reduce your development costs and simultaneously launch new applications on Windows and UNIX. For an evaluation copy of MainWin XDE, call us, send email to democopy@mainsoft.com, or visit our website. A few seconds with us might save you months of hard work.

1-800-MAINWIN

(continued from page 30)
of the *grow* function applied to the user argument. With this definition, you can write *spin1* more simply; see Listing Thirty-Five. Another use of *withSpinner* is to make the color vary in hue and use the value from *grow* to determine the time-varying speed of rotation, so that the mouse buttons cause the turning to accelerate and decelerate (see Listing Thirty-Six).

In addition to visible geometry, you can add lights to a 3D model. In Listing Thirty-Seven, you combine a white sphere, which is visible but does not emit light, and a point light source, which is invisible but emits light. You color the sphere/light pair white, shrink it, and give it motion. For convenience, you express the motion path in terms of spherical coordinates, saying that the distance from the origin of space (which is also the center of the teapot) is always 1.5 units, the longitude is π times the elapsed time, and the latitude is twice

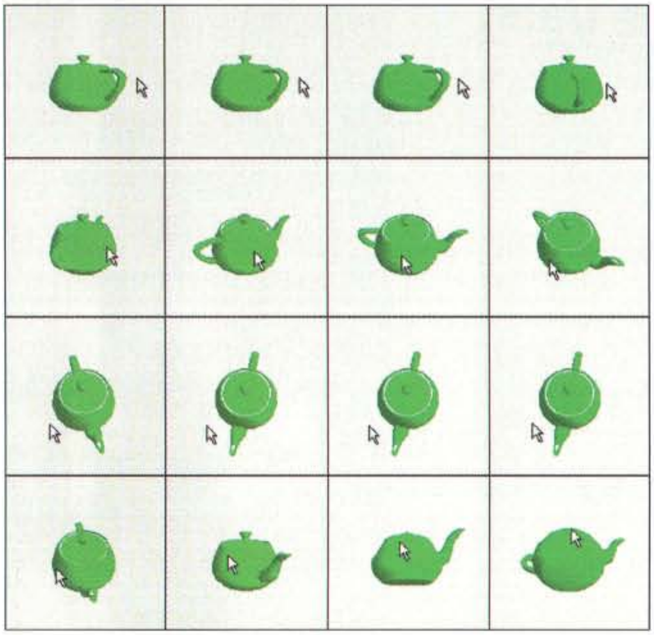


Figure 14: *Applying* mouseTurn *to a green teapot.*

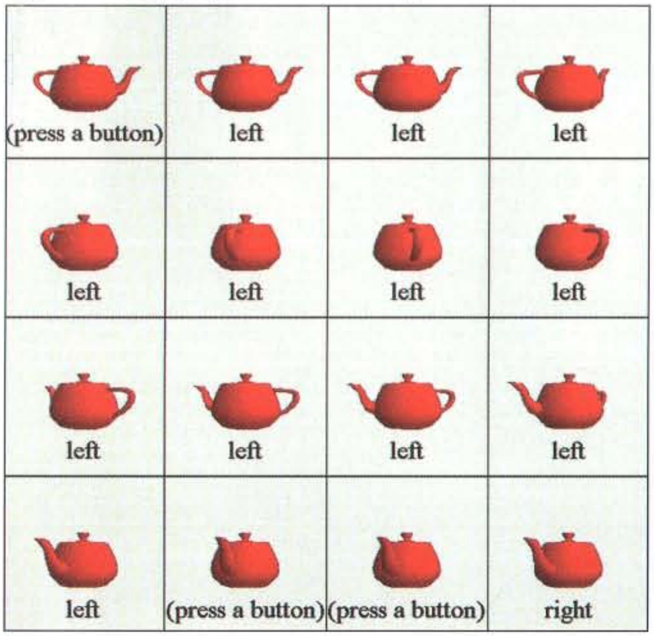


Figure 15: *Making teapots spin by controlling the rotation angle with the* grow *function.*

π times the elapsed time. Consequently, you get a motion that meanders about, but maintains a fixed distance from the center of the teapot.

Just for fun, replace the single moving light with five. A simple change suffices, if you add *delayAnims3*—a 3D variant of the 2D *delayAnims*. As Listing Thirty-Eight shows, the difference is that in the 3D version, you use *unionGs* instead of *overs*. With this function, you make a list of five copies of the moving light (see Listing Thirty-Nine), using the predefined Haskell function *replicate*, stagger them in time with *delayAnims3*, and combine them with a green teapot. Then slow down the animation to see it more clearly.

In Listing Forty and Figure 16 (a moving trail of colored balls), you define a single ball having a spiral motion, which traces the surface of an unseen sphere of radius 1.5 with a longitude angle changing ten times as fast as the latitude angle (five versus one-half radians per second). From this one moving ball, you make ten balls, each a differently colored version, and then stagger them in time with *delayAnims3*. The coloring function *bColor* produces evenly spaced hues.

As a final 3D example, Listing Forty-One presents another spiral. This time you form a static spiral, then turn it about the z-axis.

Listing Forty

```
spiral3D = delayAnims3 0.075 balls
  where
    ball   = move3 motion (stretch3 0.1 sphereLowRes)
    balls  = [ withColorG (bColor i) ball
             | i <- [1 .. n] ]
    motion = vector3Spherical 1.5 (10*time) time
    n      = 20
    bColor i =
      colorHSL (2*pi * fromInt i / fromInt n) 0.5 0.5
```

Listing Forty-One

```
spiralTurn = turn3 zVector3 (pi*time) (unionGs (map ball [1 .. n]))
  where
    n = 40
    ball i  = withColorG color (
                move3 motion (
                 stretch3 0.1 sphereLowRes ))
     where
       motion = vector3Spherical 1.5 (10*phi) phi
       phi    = pi * fromInt i / fromInt n
       color  = colorHSL (2*phi) 0.5 0.5
```

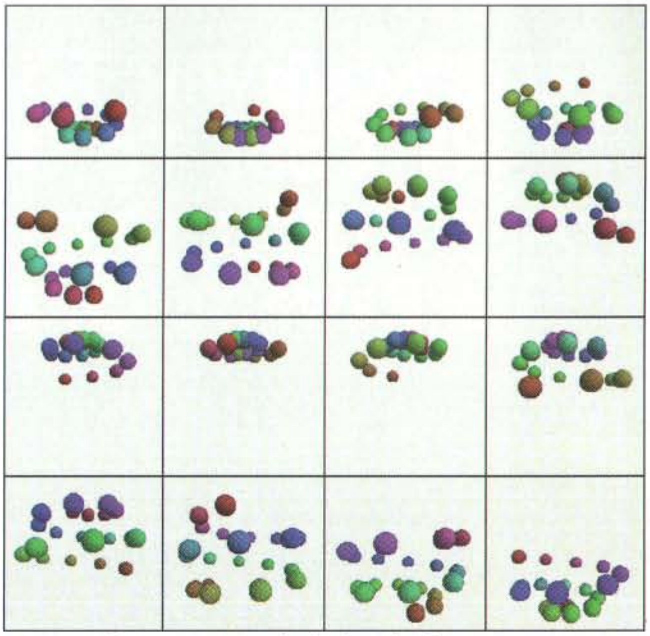


Figure 16: *A moving trail of colored spheres.*

Related Work

My interest in functional animation originally started with Kavi Arya's "A Functional Approach to Animation," *Computer Graphics Forum*, 5(4):297-311 (December, 1986). Although elegant, Arya used a discrete model of time. The TBAG system, on the other hand, used a continuous time model, and had a syntactic flavor similar to Fran's; see "TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications," by Conal Elliott, Greg Schechter, Ricky Yeung, and Salim Abi-Ezzi (*Proceedings of SIGGRAPH '94* July, 1994). Unlike Fran, reactivity was handled imperatively. Behaviors were created by means of constraint solving, and updated through constraint assertion and retraction. Concurrent ML introduced a first-class notion of events that can be constructed compositionally; see "CML: A Higher-order Concurrent Language," by John H. Reppy (*Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, 1991). However, those events perform side-effects such as writing to buffers or removing data from buffers. In contrast, Fran event occurrences have associated values—they help define what an animation is, but do not cause any side effects.

For examples of DirectAnimation, see http://www.microsoft.com/ie/ie40/demos and "Adding Theatrical Effects to Everyday Web Pages with DirectAnimation," by Salim AbiEzzi and Pablo Fernicola (*Microsoft Interactive Developer*, October 1997).

For background on Haskell, see *Introduction to Functional Programming*, by Richard Bird and Philip Wadler, (Prentice-Hall, 1987), "A Gentle Introduction to Haskell," by Paul Hudak and Joseph H. Fasel, *SIGPLAN Notices*, 27(5), May, 1992, and http://haskell.org/tutorial/index.html.

For information on Fran, refer to "Functional Reactive Animation," by Conal Elliott and Paul Hudak, *Proceedings of the 1997 ACM SIGPLAN International Conference on Functional Programming* (June, 1997), or the Fran web page at http://www.research.microsoft.com/mconal/Fran.

Conclusion

For interactive animation to expand into its potential as a medium of communication, it must become much easier to program. As this article illustrates, one step toward this goal is the replacement of imperative techniques ("how to do") with declarative ones ("what to be").

There are several features I haven't explored here, including sound, smooth flip-book animation, and cropping. There are also many opportunities for improvement: more features for 2D, sound, and 3D; improved efficiency; generation of animation "software components" to integrate with components written in more mainstream programming languages; and support for distributed, multiuser scenarios.

Acknowledgments

Todd Knoblock and Jim Kajiya helped to explore the basic ideas of behaviors and events. Sigbjorn Finne, Anthony Daniels, and Gary Shu Ling helped with the implementation during research internships. Alastair Reid made improvements to the Haskell code, and, along with Paul Hudak and John Peterson, provided helpful discussions about functional animation, how to use Haskell well, and lazy functional programming in general. Becky Elliott cut out the kid pictures, which appear with the kind permission of their owners Patrick, Charlotte, Becky, and Jake.

DDJ

A Conversation with John Knoll

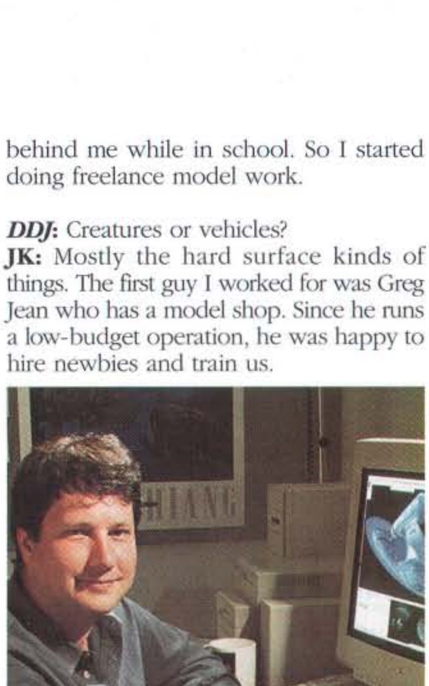
Life on the bleeding edge of computer graphics

Thomas "Rick" Tewell

As a visual effects supervisor for George Lucas' Industrial Light & Magic, John Knoll has lived on the bleeding-edge of computer graphics for over a decade. As such, he has worked on ground-breaking feature films such as *The Abyss* (which earned an Academy Award for Best Visual Effects), *Mission Impossible*, and *Star Trek VIII: First Contact*, among many others. He is currently working on the next *Star Wars* film, currently codenamed *Episode I*. In addition, John and his brother Tom are the creators of Adobe's PhotoShop image-processing software. John recently took time from his duties at Industrial Light & Magic in Marin County, California, to chat with Rick Tewell.

***DDJ*:** John, from what I understand, you transitioned from model-making into computer graphics. Can you tell us about that?
JK: Sure. When I was a kid, model-making was a hobby of mine. I got to be reasonably good at it and decided to go into visual effects as a career. I moved to Los Angeles to attend the University of Southern California film program. At USC, I tried to make contacts so that when I graduated, I wouldn't be going into an entry-level position. I was trying to get some of those entry-level-position years behind me while in school. So I started doing freelance model work.

***DDJ*:** Creatures or vehicles?
JK: Mostly the hard surface kinds of things. The first guy I worked for was Greg Jean who has a model shop. Since he runs a low-budget operation, he was happy to hire newbies and train us.



When the model was done, I'd take it out to the stage and fix things — during rigging, they'd need a hole here or something has got to move or I had to paint something to fix it because it didn't look good enough for camera. Somebody has to be around to do those sorts of things. So I would be on the stage a lot of the time when my models would be shot, which meant I got familiar with motion-control cameras. That was something that interested me. How do you get started doing that sort of thing? They didn't teach that at USC, which was mostly a live-action school. My last year at USC, I took an advanced animation class and we had a couple of manual hand-cranked animation stands. For my final project, I decided to build a simple four-channel motion-control system. This was in 1984. I bought a used Apple II and a four-channel serial-controlled CNC milling machine, which ran four stepper motors. And I bought a bunch of surplus stepper motors from C&H Sales and various bits and pieces. Although the camera got booked in two-hour blocks during the week, it was free during the weekend. Consequently, after the last session on Friday night, I could go in there, take the hand-cranks off, bolt my motors on, set up the computer, and shoot as long as I had it all cleared off by the first scheduled block on Monday. It was a lot of fun.

***DDJ*:** This was an Apple II?
JK: An Apple II Plus with a whopping 64K of RAM. I had a digital I/O board so I could control various relays.

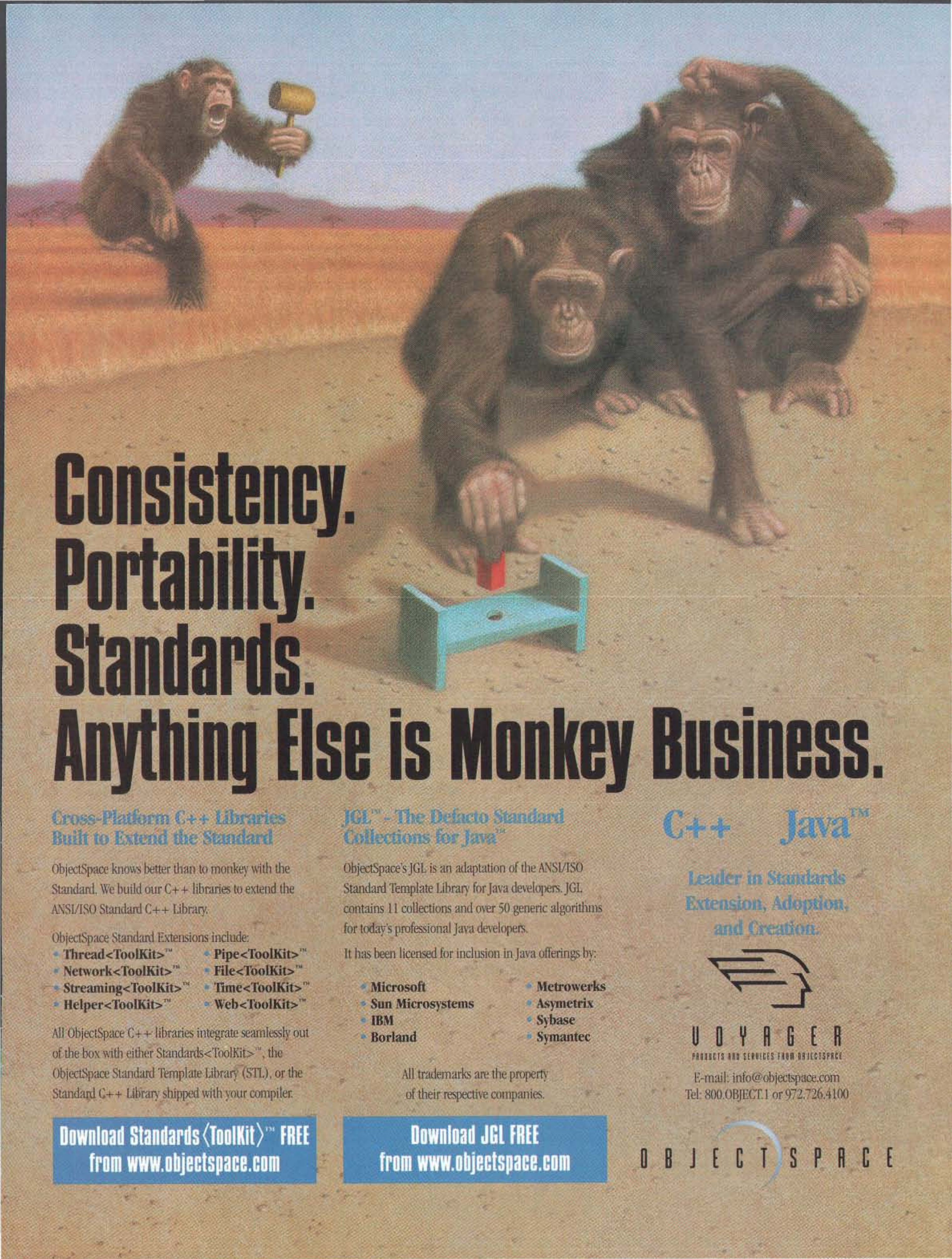
***DDJ*:** So primarily, you were using the Apple II to do the motion control, and the camera was just a regular film camera?
JK: Yeah. What I was shooting was slit scan. It was a process I read about and was fascinated with and I wanted to try it. You really need a computer to control that stuff.

***DDJ*:** Did you write the software for the Apple II?
JK: Yes.

***DDJ*:** So you were familiar with programming at that time?
JK: A little. Actually, before I started at USC (in 1980), my dad got an Apple II as part of his university research work. After dinner, he'd go work on his research but he encouraged my brother Tom and I to play with it. This was in 1978.

The wonderful thing about the Apple II was it had this Basic interpreter built into ROM, so all you had to do was turn

Rick works for Sequoia Advanced Technologies. He can be contacted at thomas .tewell@seqadvtech.com.


Consistency.
Portability.
Standards.
Anything Else is Monkey Business.
Cross-Platform C++ Libraries Built to Extend the Standard
ObjectSpace knows better than to monkey with the Standard. We build our C++ libraries to extend the ANSI/ISO Standard C++ Library.
ObjectSpace Standard Extensions include:
• Thread<ToolKit>™ • Pipe<ToolKit>™ • Network<ToolKit>™ • File<ToolKit>™ • Streaming<ToolKit>™ • Time<ToolKit>™ • Helper<ToolKit>™ • Web<ToolKit>™
All ObjectSpace C++ libraries integrate seamlessly out of the box with either Standards<ToolKit>™, the ObjectSpace Standard Template Library (STL), or the Standard C++ Library shipped with your compiler.
Download Standards⟨ToolKit⟩™ FREE from www.objectspace.com
JGL™ - The Defacto Standard Collections for Java™
ObjectSpace's JGL is an adaptation of the ANSI/ISO Standard Template Library for Java developers. JGL contains 11 collections and over 50 generic algorithms for today's professional Java developers.
It has been licensed for inclusion in Java offerings by:
• Microsoft • Sun Microsystems • IBM • Borland • Metrowerks • Asymetrix • Sybase • Symantec
All trademarks are the property of their respective companies.
Download JGL FREE from www.objectspace.com
C++ Java™
Leader in Standards Extension, Adoption, and Creation.
VOYAGER
PRODUCTS AND SERVICES FROM OBJECTSPACE
E-mail: info@objectspace.com
Tel: 800.OBJECT.1 or 972.726.4100
OBJECTSPACE

(continued from page 34)
the computer on and start typing in lines. That was a lot of fun. I feel privileged that one of my first exposures to computers was when they were so simple. There was only so much that these really primitive computers could do, so it didn't take a lot to kind of learn everything there was to know about them. As the computers became more complicated, you could learn gradually. I can only imagine what it's like to dive into what programming is like now. I've had 20 years of exposure to it. Today, it's incredibly complicated for somebody just coming out of school.

***DDJ*:** At what point did it occur to you that the computer could actually be a tool for more than motion control or camera control—that the computer could actually be used to generate computer images suitable for film?

JK: A lot of people saw it coming. I read about computer graphics and had friends who were members of SIGGRAPH so I saw the tapes, and was fascinated by it. It wasn't really interesting enough to me at that point in the early '80s. I thought it was neat but not ready for feature films. But then as it started getting close to being ready, I became one of the first people pushing for it. I was computer graphics designer on *The Abyss* [circa 1989], which was one of the first realistic pieces of computer graphics in a feature film. At least that was our intent.

***DDJ*:** When did PhotoShop come into play?

JK: Actually, it was somewhat accidental. As I said, when I was a kid, one of my hobbies was model making. I got to be fairly good at that and it got me into the industry. But when model making turned into a profession, it sort of killed it as a hobby. It's not much fun to build models all day, then go home and build more models.

Since I was interested in motion control, I got a computer and started building motion-control systems for it. That became my new hobby. Because I knew people who were shooting motion-control elements with the models I was building, I began getting work as a camera assistant on motion-control stages. Then I got hired as a motion-control camera assistant at Industrial Light & Magic (ILM). Pretty soon I was doing motion control full time and its appeal as a hobby was greatly diminished.

I started at ILM in 1986 and had just gotten a Macintosh, my first sophisticated computer, and started writing little graphics programs as my new hobby. ILM was the first place I ever worked that had a computer-graphics department and, when I wasn't working in motion control, I'd go there to see what they were up to. They had this laser film scanner, where you could scan in a piece of negative and generate a digital image. They had the Pixar Image Computer, a nice high-quality frame buffer where you could do manipulations to a picture and film it back out. I had a demo of something so trivial now, you hardly even think of it. This guy brought up an image on the screen and simply sharpened it. That actually seemed miraculous at the time and made a huge impression on me.

Visual Effects Supervisor John Knoll (left) working with Senior Model Maker John Goodson (right) on a helicopter from Mission Impossible.

About that time, my brother Tom was at the University of Michigan working on his doctoral thesis. He had pursued computer programming much more seriously; that's what he had wanted to do for his career.

He was trying to get his doctorate in computer vision and the first part of any computer vision stuff is image processing. He was doing his thesis work on a Mac Plus and writing these image-processing algorithms as MPW shell tools. That was much like how Pixar Image Computers worked. You typed in command-line arguments from a UNIX command line to run C-shell scripts from the Sun to control the frame buffer on the Pixar. That was sort of the same thing Tom was doing on his Mac.

I saw a lot of the similarities. Then the Mac II came out. It had a math coprocessor. It had color. It was faster. It had more memory. I had to have it because I thought it was so neat. When that machine first came out, displaying a color image on it from a programming standpoint was a big deal. I wasn't terribly interested in the mechanics of the palette manager, window manager, and all the things that were required to display a color picture. What I was interested in was the code that figured out how bright a pixel should be. One of the hobby things I was doing was writing a little ray tracer. Tom told me to do the math, figure out how bright the pixel ought to be, and just write it to disk as a raw image. He said I could use his tools, which could read a raw block of bytes on the disk and display it as a picture and do various transforms to it.

I did this for a while, but it was cumbersome and I thought what would be neat was if we just built the display portion of this into an application so that I wouldn't have to fire up the whole MPW thing and run the shell tools to do this. One weekend, Tom spent a few hours bundling some of those functions in to this program called "Display." Once he had that working, I started bugging him for more stuff. It was like nothing was ever good enough. So we started adding more features until it struck me that we should sell this. We could get an ad in the back of *MacWorld* and sell it for 50 bucks. Tom was really skeptical.

***DDJ*:** Did you ever sell the product?

JK: No. Mostly what Display did was conversions. We had gotten it so that it could read several different image file formats. You could write several different image file formats and there were a couple of things you could do to them in the meantime. You could convert a color image to black and white.

I was completely full of naïve optimism. I showed it to a friend of mine at SuperMac, which was in alpha with a program called "PixelPaint." SuperMac was seriously considering making us an offer to bundle Display with PixelPaint as a file-format conversion utility. They had already run all their spreadsheets about how many units they thought they were going to sell of PixelPaint and what kind of deal would they want to make with us on bundling this. That added up to a number that seemed like this was worth doing.

I called Tom and said SuperMac was interested, so he scheduled two days a week to work on it full time. After two or three months, it really did a lot of things. It didn't really fit in my mind as utility any more. It was a program in its own right that wanted to be sold as its own product. One day I called Tom up and told him that I didn't think there would ever be an opportunity like this like thrown at our feet again. We just had to drop everything to make this happen.

Tom estimated he was six months from finishing his doctoral thesis. In a supreme act of faith, he stopped working on his thesis and started programming full time. We greatly underestimated how much work this was going to be. When Tom

(continued from page 36)
stopped school, he figured he had about six months of programming and we'd wrap up Version 1.0 of this program, and he could start next semester and finish his thesis. Meanwhile, we'd be making some money.

From the time he decided to stop school until Version 1.0 shipped was almost two years. It became much bigger than we thought it would, but it kept getting better and better. Tom is really a superb programmer. He's one of the best engineers I know. He just wrote this terrific, great code.

At the time, I moved from motion control over to computer graphics, so I was doing a lot of work on the Pixar Image Computers—running composites and doing image-processing scripts. That drove a lot of my input as to what kind of features ought to be in PhotoShop. I would try to do more and more of my work in PhotoShop and try stuff. That's sort of how "feathering" got born. It was actually me using it for little projects that helped define the feature set.

Version 1.0 was a usable tool largely because I was trying to use it to solve real-world problems. I would run into something that would just stymie me. There's got to be a way of doing this, and then Tom would scratch his head and go, "That would be hard." He would think about it for a while. I would talk to him a few days later and he would say, "I was thinking about that and I had this great idea."

I was goading him a little bit, too. I would say, "You know what I really want to do? I want to make one of these selections so that I can like select some area and then the paint only affects just the area selected." Tom would say, "Oh, that's going to be impossible to make that go real time. It's going to be really slow." I'd say "Oh, come on, Tom. I'll bet you can do that." About a week later he would say, "I was thinking about it, and I think I've got a way." It was often a whole lot of exchanges like that where at first Tom thought it would be really hard, but he would keep thinking about it. He's brilliant that way, and he would come up with a clever solution to the problem.

***DDJ*:** When PhotoShop was born, the industry was in some interesting transitions in computer graphics.

JK: Yes. We started on PhotoShop in September of 1987. I think 1.0 shipped in January of 1990. There was some time between when we started and when it shipped. A lot of things happened in that time. I started working in computer graphics…it wasn't until late 1988, I think. The first thing I did in computer graphics was a Pacific Bell Smart Yellow Pages commercial.

***DDJ*:** With a Pixar?

JK: Yeah. A Pixar Image Computer is basically a frame buffer. Lucas Film Computer Division was working on what became the Pixar computer. "Pixar" adopted that name as the name of the company after George [Lucas] sold it to Steve Jobs.

***DDJ*:** So that was something that was invented and not available anywhere else except for here?

JK: Right. We had two of them here that we used for composite work and various image-processing things. On all the old Pixar films like *Andre and Wally B* [circa 1984], they would render different parts of the shot as separate passes so the character in the foreground would be rendered separate from the background. Then they would composite them together, and the tool they used to do it would be the Pixar Image Computer.

We try to use off-the-shelf software wherever we can

***DDJ*:** When *The Abyss* was created, what was the state of computer graphics?

JK: In general, no one thought of computer graphics as something you could use for real on a feature film to do something that looked realistic. The one exception was the stained-glass man [from *Young Sherlock Holmes*, circa 1985], which was a pretty remarkable achievement, and it's the only thing that had ever been quite like that to that point. Stuff like *Last Star Fighter* [circa 1984], nobody really considered realistic. But I was impressed with stained-glass man because it had things like depth of field.

Right after I started, our computer-graphics department had done this *Star Trek IV* [circa 1986] dream sequence with the floating heads. It didn't look very realistic. It was intended to be a stylized thing. I don't know if anybody thought that our tools in house were ready to do something super realistic.

I remember we got the storyboards on *The Abyss*, they were these beautiful shaded drawings. They are really fascinating. The imagery was really neat. "Wow, these are going to be really cool shots—whoever does this and however it gets done." A lot of different approaches were being bandied about with things even as weird as stop-motion animation with clay with images of water projected onto it. Things that almost certainly never would have worked.

We had just gotten an SGI with Alias, and Jay Riddle in the computer-graphics department did a little test making some sort of a water tentacle thing. It was not a sophisticated test, but he did it really quick. He did it, I think, overnight and showed it to Jim Cameron [*Titanic* writer/director] the next day. Jim was really surprised how quickly that had been done because the reputation was that computer graphics was really, really slow and very expensive and the complete antithesis of interactivity. You'd talk to these guys and they'd disappear for months, and then they would come back with something you didn't want. "I want it to be more like…" "Well that will be another six months."

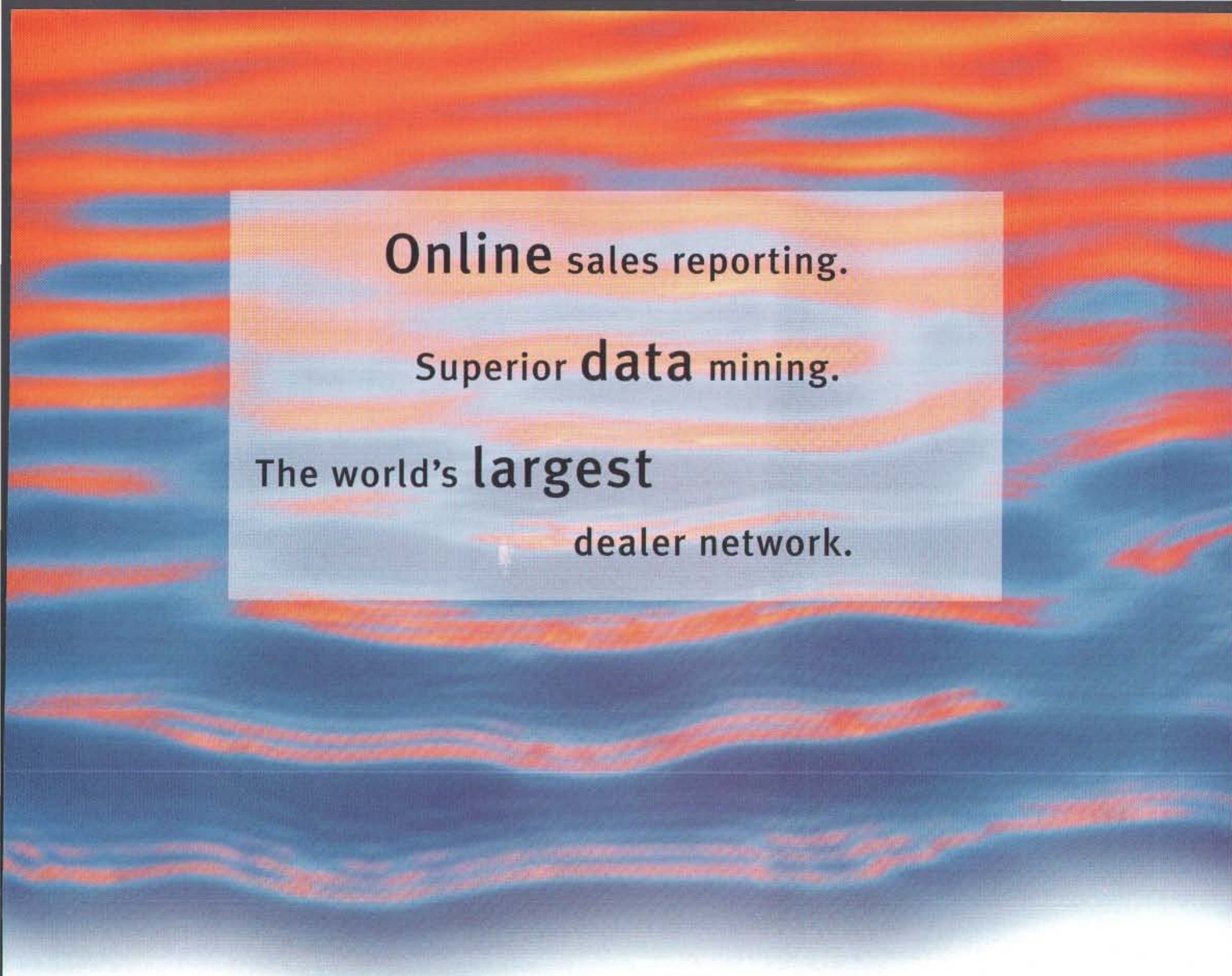
***DDJ*:** But they felt this was an intricate part of the film?

JK: Jim's position was that if the water tentacle sequence—while it was a bold thing to attempt—didn't work or ended up looking terrible, he could cut it out of the movie and he could still make the movie. He wasn't hinging the success of this picture on this effect working. It was only like 25 shots. This seemed like a huge number to us at the time, but it is hardly anything now. So we started this R&D project to do this thing, and we wrote a bunch of new software to do it. We switched over from Rays to RenderMan, which Pixar had just gotten going.

***DDJ*:** There was nothing on the street that could do this at the time?

JK: No. We used the RenderMan renderer but we wrote custom shaders to do the fake refraction and get the right amount of reflection for fog and that sort of thing. We had to write the software for it to do the rippling of the surface and to "skin" it. The way it was actually done was, we animated a spline in space—a 3D path—and we had a bunch of cross sections. They were animated separately, so it was just a bunch of circles, and we scaled them. And then, there was a piece of software called "Skin" that would take all of the circular cross sections and place them perpendicular to the spine at particular points and skin the surface.

Then there was another program that would let you place a bunch of 3D noise generators in the world, and it would take the patches and subdivide them into smaller patches and perturb all the control



Just another day on the river.

When it comes to selling software online, we do more in a day than most are equipped to accomplish in a year. That's because no one approaches the business like we do. We offer the essentials: commerce site management and product distribution, and also provide complete online sales reporting, database analysis, advertising and marketing support. Our patent-pending technology delivers absolute reliability, with features that include automated load balancing and redundancy. And perhaps the best part? All of this is managed for you, with a staff working 24 hours a day, 7 days a week to oversee your site and keep your customers happy. So if you're looking for an Internet marketing strategy that replaces problems with profits, go on the river today. For more information, call 1-800-207-2755 or e-mail us at marketing@digitalriver.com.

DR
Digital River

(continued from page 38)

vertices according to the sums of all the sine waves from 3D noise generators. So the model was created new per frame based on this program, so some work was involved. How do you do motion blurs when you're actually just changing the model from frame to frame without taking one model and moving it? Some hacks were made. Actually it's the same model, but what we're doing is we're moving these vertices from here to here. You would write two-root files. They contain all the same CVs [control vertices] and then there was a script called JR2R that would take the two-root files and make them look like it was one model just moving from this frame to that frame.

***DDJ*:** Then comes *Terminator 2* [circa 1991], which has something (not quite like the water tentacle) but it has the Mercury guy and that was from James Cameron.

JK: Yeah. Jim said it was a big gamble. If it didn't work, he could always cut it out of the picture, but based on his experience on *The Abyss*, he went much bolder on *Terminator 2* with making a character that had to be done with computer graphics. And it had to work because if you cut that out of the movie, you've got nothing left. All the things of being able to change shape from this to that and to melt and then reform itself. Well, the effect has to work or you don't have a movie. Yeah, it was a sign of his faith in the technology.

***DDJ*:** In *Jumanji* [circa 1995] we have the first computer graphics hair that actually flows and moves, and the depth is there, and it is so stunningly realistic that it was actually an amazing achievement for computer graphics. Did that require custom tools or was there a point where you could actually use off-the-shelf components to actually do this?

JK: We try to use off-the-shelf software wherever we can, but a lot of things we're called upon to do just can't be done with off-the-shelf software. So we have a pretty good size software-development staff just to develop these tools; otherwise, we would just have to say, "No, we can't do that."

***DDJ*:** Do you still do that today?

JK: Yeah.

***DDJ*:** Do producers come in and say, "We want to produce a film and here are the special effects that we want" and you just go, "I don't think so."

JK: Well, no. We gulp and say, "Okay, we can do that. Here's the budget." Then they gulp.

You can usually spend your way out of just about any hole there is. If you put enough time and man hours into something, there's usually a way to do it and I can think of very few exceptions where we just have to give up and say, "No, that just can't be done." There are some things that would be extremely difficult and we could never do realistically, at least not yet. But most of the things we're asked to do are at least within some amount of R&D of what we're capable of. George [Lucas], on this new *Star Wars* picture, wrote a lot of things into the script without worrying about how the hell are we going to do this. He just writes things he thinks are neat.

***DDJ*:** Martin Hash has created a product called Animation Master and is trying to make a film, *Telepresence,* for $2 million which positively could not be made for $2 million if a studio did it based on the effects he wants to put in there.

Do you see a trend coming where independent filmmakers can use off-the-shelf components to actually have "big budget" special effects in films? Up to now, independent films have been pretty much lacking special effects that are just sort of character driven.

JK: It's already happening. A bunch of friends of mine are starting up these garage operations—little one-man digital facilities—and they do things for TV shows or low-budget features. They're able to do the kind of work now just at home with PCs. It used to be that you had to have the whole full-blown production mechanism here for it, and now you can do some pretty good looking stuff.

***DDJ*:** Like Electric Image?

JK: Yeah. With Electric Image, After Effects, and PhotoShop, you've got a little production facility there.

***DDJ*:** Speaking to a peer programming audience, what do you see as the next generation of products for computer graphics?

JK: Well, I don't think there are any real specifics that are easy to predict. But I think the general trend is to try and eliminate as much machinery between the artist and the art as possible.

One of the things that has been really liberating about moving to digital-production techniques is that it used to be that huge amounts of effort went into just the mechanics of not getting the matte line or not getting the wrong color in a shot, for instance. That's where a lot of your energy went—just trying to get rid of the really

(continued from page 40)
obvious things. Now, you can take that stuff more for granted. Today, an artist spends more time working on the aspects of the work that make the shot look good or not look good, and not so much on the mechanical. I see that trend continuing.

Right now, my biggest complaint about the way that a lot of these digital tools work is that they're still kind of awkward, and the artist spends too much time working on things that have nothing to do with the shot looking good or not. It's editing exclusion lists and making sure your aliases are pointing to the right directories. There's a lot of machinery that the artist still has to deal with that, as software gets better, they're going to spend less of their time of doing and more of their time focused on the real art of it.

***DDJ*:** What about these new digital interfaces like FireWire? Do you see that again liberating artists so that digital images can go straight into the machine?

JK: I think that all these technologies like this are wonderful. I spend a lot of my time living on the bleeding edge, where we're just trying to get something done almost no matter how painful it is. We work with these kind of kludgy custom-written things that just barely work well enough to get through the shot or you really wouldn't want to do that a whole lot more. And what happens is that like five years down the line, the commercial applications end up with a lot of functionality that we have very painstakingly hand crafted—like morphing, for example.

Back on *Willow* [circa 1988], Doug Smythe spent time writing the first morphing program that worked well for what we did and let us do these shots that were sort of impossible otherwise. We made good use of it. I used it on *The Abyss* to actually do the face animation with morphing. We used it on *Terminator 2*. Then Elastic Reality hit the market and once that capability was present in the commercial program, it was at least as good as our morph program. In some ways, it was better, and there was no reason to keep working on our program.

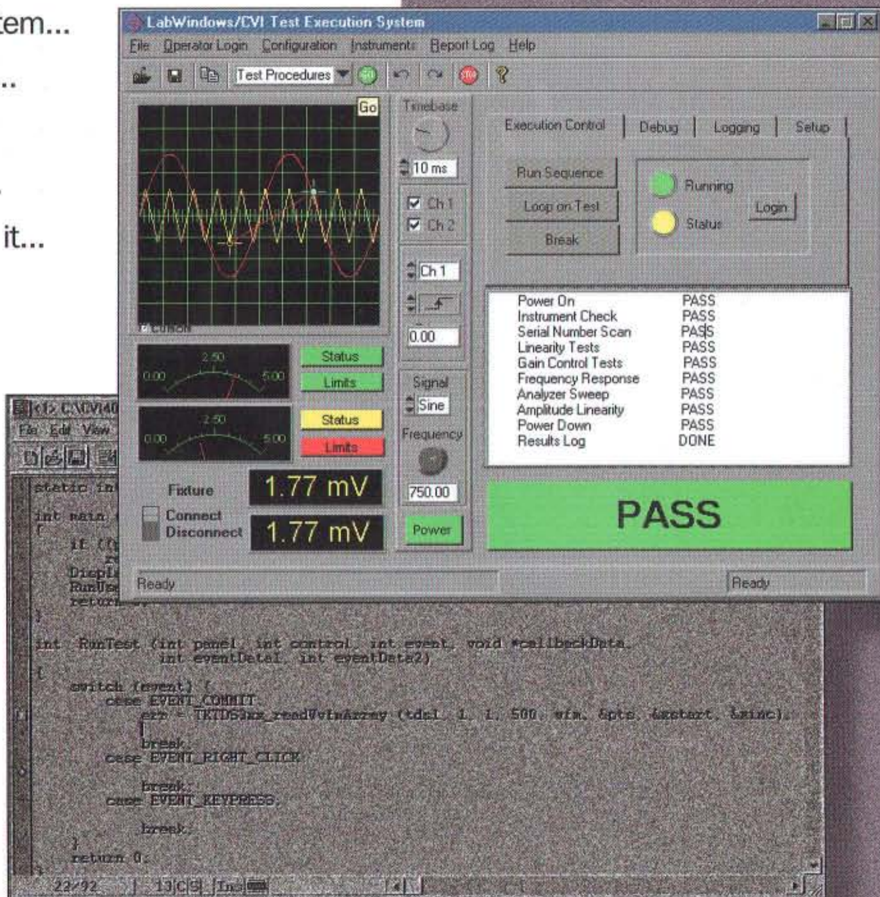
A commercial application now had the same functionality, and you could buy it for nothing. That's a good example of something that we sort of suffer through getting the first version, and then people see the results of that, and they go, "Oh, man, I want this." So a bunch of commercial developers can jump in and say, "Well we can provide that." They write a good interface on it, on something that's actually debugged with appropriate error messages and all those kinds of things that commercial software brings to the equation. And then it's available to everybody.

DDJ

Build on a Rock Solid Test Foundation

LabWindows™/CVI
Virtual Instrumentation Tools for C/C++

When you need to build a BIG test system...
When you can't afford to get locked in...
When it has to be fast...
When you need a solution that works...
When your production line depends on it...
When your job is on the line...



Why look anywhere else?

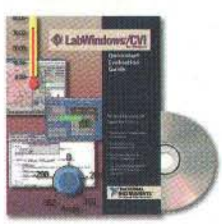
- ✔ Standard ANSI C Programming
- ✔ Lightning-fast code generation
- ✔ Open development environment
- ✔ VXI*plug&play* driver standard

For data acquisition, GPIB, and VXI instrument control, analysis, and user interface tools, LabWindows/CVI has everything you need under one roof.

Put LabWindows/CVI to Your Test!

NATIONAL INSTRUMENTS™
The Software is the Instrument™

U.S. Corporate Headquarters
Tel: (512) 794-0100 • Fax: (512) 794-8411
info@natinst.com • www.natinst.com
Worldwide network of direct offices and distributors.



Call today for your FREE LabWindows/CVI Evaluation Version
(800) 661-6063
(U.S. and Canada)

A Windows 3D Model Viewer for OpenGL

Combining Win32 with OpenGL

Jawed Karim

OpenGL is known in the UNIX world as the 3D API behind high-powered scientific applications. It has recently gained attention in the PC sector, thanks to the computer-game industry, which has embraced OpenGL as an API standard for 3D game programming. Furthermore, 3D hardware acceleration for PCs has extended the range of applications for OpenGL even further.

The OpenGL API is intuitive, easier to use, in my opinion, than Microsoft's Direct3D API, and is portable among platforms. In this article, I'll present a model viewer for use with OpenGL on Windows 95/NT. First, however, I'll describe the important parts of a Quake2 model viewer—an OpenGL-based system written in C/C++—that displays wire-frame and texture-mapped models (see Figure 1) from Quake2 and provides a basic interface to modify their appearance. In the process, I'll focus on file formats (MD2 files for models, and PCX files for textures), passing the data contained in the files to OpenGL for rendering, and interfacing Win32 with OpenGL using an API called "WGL." The archive Q2M-SRC.ZIP contains the Quake2 Model Viewer source code, while Q2M-BIN.ZIP is the Quake2 Model Viewer EXE file. Both are available electronically; see "Resource Center," page 3.

Jawed studies computer science at the University of Illinois at Urbana-Champaign. He works part-time at the National Center for Supercomputing Applications, and can be contacted at jkarim@students .uiuc.edu.

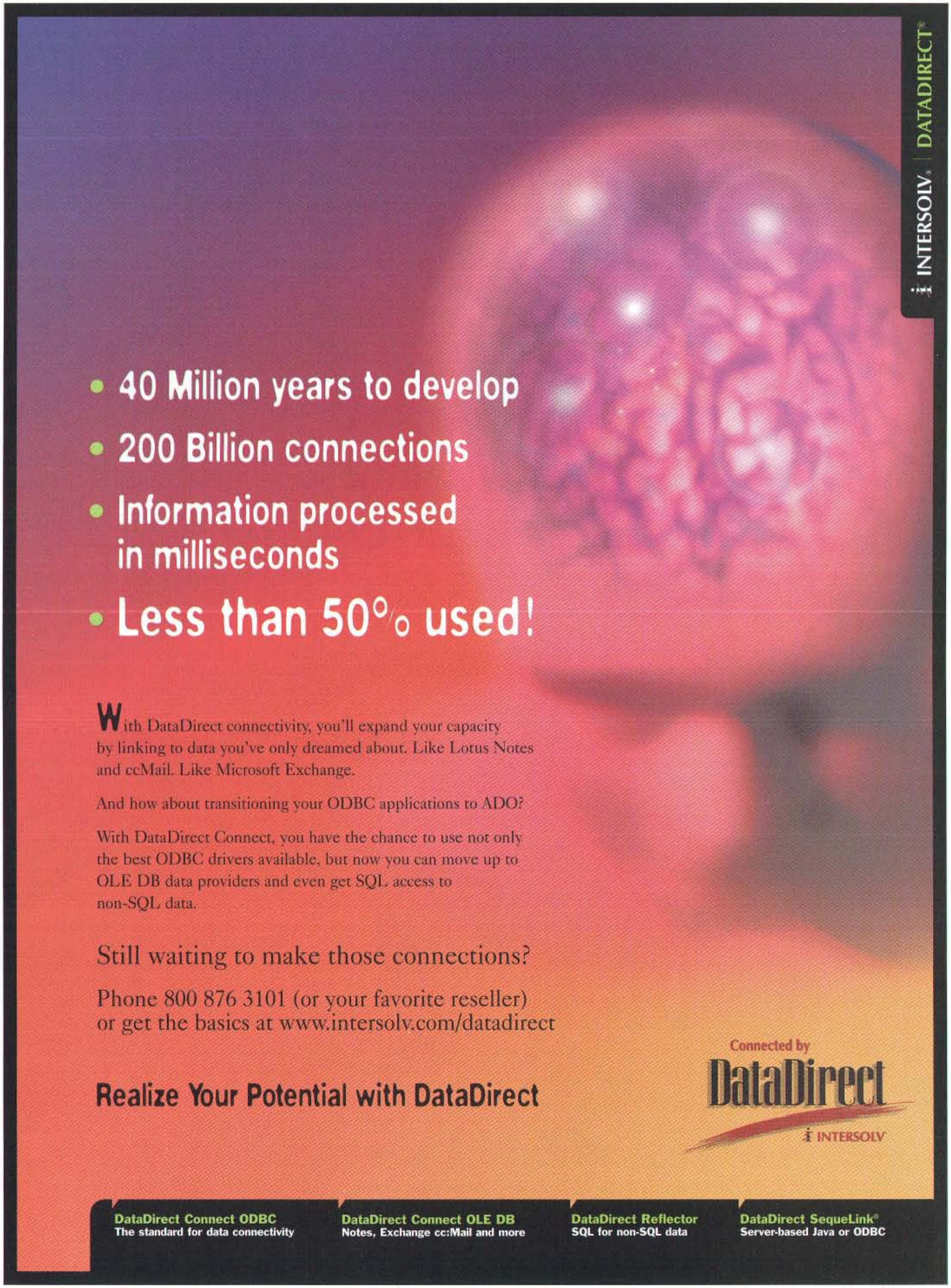
Reading the MD2 File Format

The only official source of information about Quake2's MD2 format is code by John Carmack of id Software; this code writes 3D polygon mesh data to an MD2 file (available at ftp://ftp.idsoftware.com/). Anyone who has looked at this source code will notice that some of the *structs* in Quake2 Model Viewer's md2.h (available electronically) are derived from it. Writing the MD2 reader basically involves converting John's code from reading MD2 files to writing them. Figure 2 illustrates the binary structure of an MD2 file.

To display the textured Quake2 models, four specific types of information are needed (see Figure 3):

- 3D vertex coordinates.
- A list of triangles consisting of those vertices.
- 2D texture vertex coordinates (one for each 3D vertex).
- The texture image.

All of the 3D vertices in the model are stored in one array. When the triangles (which are made up of those vertices) are defined, all that has to be stored for each vertex of a triangle is an index number to the big vertex array. The reason for this is simple: Since many of the vertices are shared between triangles, storing each vertex once saves memory. In addition, linear transformations can be


INTERSOLV | DATADIRECT®
• 40 Million years to develop
• 200 Billion connections
• Information processed in milliseconds
• Less than 50% used!
With DataDirect connectivity, you'll expand your capacity by linking to data you've only dreamed about. Like Lotus Notes and ccMail. Like Microsoft Exchange.
And how about transitioning your ODBC applications to ADO?
With DataDirect Connect, you have the chance to use not only the best ODBC drivers available, but now you can move up to OLE DB data providers and even get SQL access to non-SQL data.
Still waiting to make those connections?
Phone 800 876 3101 (or your favorite reseller) or get the basics at www.intersolv.com/datadirect
Realize Your Potential with DataDirect
Connected by DataDirect
INTERSOLV
DataDirect Connect ODBC
The standard for data connectivity
DataDirect Connect OLE DB
Notes, Exchange cc:Mail and more
DataDirect Reflector
SQL for non-SQL data
DataDirect SequeLink®
Server-based Java or ODBC

(continued from page 44)
performed on the entire array at once, thereby speeding rendering time. Since the texture image itself is not a part of the MD2 file, it can be read in from a conventional PCX file.

Before starting, you must know how much data to expect. The file's header section tells you the number of vertices, triangles, and texture coordinates contained in the file. Knowing when to stop, you can go into a loop and read the information in chunks. To store all the data, use the vertex structure in Listing One (listings begin on page 96).

Each triangle is defined by its corners, *a*, *b*, and *c*. These values are indices to an array of type *make_vertex_list*, which is a list of all 3D vertices in the entire model. The remaining six integers represent the 2D texture coordinates for every vertex. Listing Two is an example of a structure for holding this data. Using such a structure, the coordinates of the three vertices of the first triangle in the model can be referenced (see Listing Three).

In a Quake2 model, the only things that differ from one frame to the next are the 3D coordinates of the triangle vertices; the vertex indices and texture coordinates remain the same. From frame to frame, each triangle still consists of the same three vertices— only the vertices undergo linear transformations. To hold each frame in an array, you create another array of type *make_frame_list* (Listing Four), each of which contains an array of vertex coordinates (Vertex 1, 2, and 3, respectively). There exists one copy of this array for each frame. Having filled all of the data structures, you can look up the coordinates of any polygon in any frame; see Listing Five (the coordinates of polygon P in frame F).

Figure 1: *The 3D model viewer in action.*

Texturing the Object

Quake2's model textures reside as separate PCX files, either in the pak0.pak file or quake2/baseq2 directory. Since OpenGL itself does not provide a way to read the binary PCX graphics file format, you can read the PCX file and pass its data to OpenGL.

Figure 4 describes the PCX format. The three basic sections in the file are the header, pixel data, and palette data. You can use two arrays of type *unsigned char* to store the last two sections. The header contains some basic information about the particular file, such as the PCX version, and the file dimensions. If the file is actually a PCX Version 5 file, the first two bytes in the file must be equal to 10 and 5, respectively. Having determined the image dimensions from the header section, you dynamically allocate an array of type *unsigned char* of *size(width*height)* for the pixel data and read it into the buffer byte-by-byte. Because a Version 5 PCX file can support exactly 256 colors, the size of the palette section is always 768 bytes (3*256, or RBG*256).

When the *CImage::Read (char filename[])* function is finished, the *m_pixel_buffer* array is filled with all the pixels in the image, and *m_palette_buffer* contains consecutive RGB values for each of the colors.

How do you get the color of a specific pixel in the image? The pixel buffer simply contains index values of the palette buffer. Listing Six shows two methods. The R, G, and B components of the first pixel (pixel zero) in the image are Listing Six(a). However, because the palette array contains consecutive RBG values (RGBRGBRGBRGB…) for all the colors, the individual R, G, and B values at pixel position P are obtained by properly offsetting the array index; see Listing Six(b).

```
modelheader <dmdl_t>
{
  int num_frames;
  int num_vertices;
  int num_tris;
  ..
  ..
}
```

The modelheader is a structure of type dmdl_t. It contains values describing the amount of data contained in the file.

```
g_skins <char [MAX_LEN][64]>
g_skins[0]
g_skins[1]
..
gskins[63]
```

Skin names (not used).

```
base_st <dstvert_t [MAX_VERTS]>
{
 short s;
 short t;
}
```

An array of structures of type dstvert_t contains texture-mapping coordinates for each vertex.

```
tri <dtriangle_t [MAX_VERTS]>
{
 short index_xyz[3];
 short index_st[3];
}
```

The triangle structure holding array indexes for the two arrays containing all vertices and texture-mapping coordinates.

```
buffer <dalias_frame [MAX_VERTS]>
{
 float scale[3], translate[3];
 drivertx_t verts[3];
}
```

This is the actual structure containing the 3D vertex xyz coordinates.

Figure 2: *The binary structure of an MD2 file.*

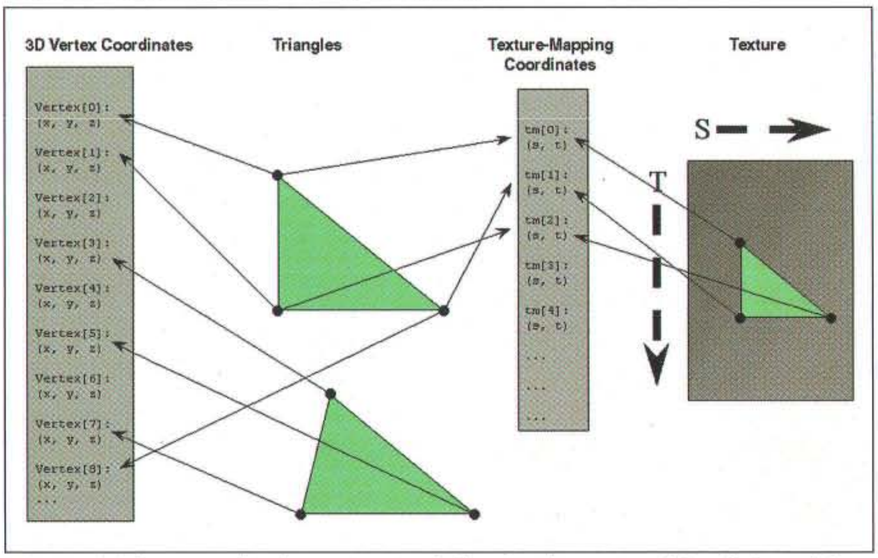


Figure 3: *The types of information needed to display textured Quake2 models.*

Finally, to be able to reference color values at specific (X,Y) coordinates in the texture, *P* is substituted by *X+Y*Width*, where *Width* is the width of the texture; see Listing Six(c).

OpenGL

Once the necessary data is organized and stored in memory, you can start rendering using OpenGL. But first, some of OpenGL's texturing options must be set. In particular, you must specify how to treat textures when wrapped and indicate the "minification" and magnification filters (Listing Seven).

In addition, back-face culling and texturing have to be explicitly enabled. Since you won't be looking at the backsides of polygons, you only have to enable front-side filling of polygons. Lastly, you specify the texture function (Listing Eight).

OpenGL's *glTexImage2D()* is the function that actually textures the object. It expects to be passed, among other parameters, a pointer to an array containing successive RGBA values for each pixel in the texture (for example, RGBARGBA-RGBA...).

Thus, before calling *glTexImage2D()*, two changes must be made:

1. The pixel and palette data read from the PCX file must be copied into another array, of a format that *glTexImage2D()* can accept as a parameter.
2. Because OpenGL requires the dimensions of a texture to be powers of two, the texture has to be rescaled first using *gluScaleImage()*.

Both of these steps are accomplished in *CImage::Image2GLTexture()*, which first creates a new array called *unScaled*, fills it with RGBA components, and rescales it to an appropriate size. The loop in Listing Nine fills a new array with RGBA components of each pixel in the texture, again offsetting the array indices as in the PCX code.

> *WGL provides an interface between the Win32 API and OpenGL*

Now the texture contained within *unScaled* can be rescaled to have dimensions that are powers of two. To prevent the texture from losing much quality while keeping the performance at a reasonable level, a power of two that is closest to the original dimension will be used. For example, if the original width is greater than 256 pixels, the new dimension should be 512 pixels. If the original width is 128 or greater (but less than 256), the rescaled dimension should be 256. After a series of *if* statements have determined a good fit for the new dimensions, a call to *gluScaleImage()* rescales the texture (Listing Ten).

Finally, the *glTexture* array can be passed to OpenGL as follows: *glTexImage2D(GL_TEXTURE_2D,0,4,scaledWidth, scaledHeight,0,GL_RGBA,GL_UNSIGNED_BYTE, glTexture);*. Table 1 provides a quick explanation of the parameters.

Creating an OpenGL Rendering Context

WGL provides an interface between the Win32 API and OpenGL. It sets up a palette for your rendering window and handles such things as double buffering. To do this, you usually need to use four or five of the fewer than 20 WGL functions. I have written a basic C++ wrapper class for the functions that is easy to use. Most of the code in the *COpenGLWindow* class is taken from Silicon Graphics' OpenGL Developer Tools CD-ROM for Windows 95/NT, which interestingly has become a collector's item since SGI's "Fahrenheit" deal with Microsoft. (SGI is cooperating with Microsoft on the next generation of OpenGL. Since the agreement, SGI's, OpenGL drivers for Windows 95/NT have disappeared from the SGI web site, and the SGI OpenGL Developer CD-ROM for Windows 95/NT is hard to come by. However, there are several web sites mirroring its contents, including http://jawed.ncsa.uiuc.edu/.)

The dimensions of the rendering window are passed to the constructor, but its window handle must be passed to the *OpenGLWindow::Create()* class member function to actually create the rendering context.

WGL does not physically create a window for you; that is Win32's responsibility. WGL creates an OpenGL rendering context for a window that has already been created. If you want a window to create and destroy its OpenGL rendering context as the window is created and destroyed, simply catch the WM_CREATE and WM_DESTROY messages in the window's window procedure. Then call *OpenGLWindow::Create()* and *OpenGLWindow::Destroy()*, respectively, as has been done in inter.c's *GraphicsProc* function (available electronically). The only other time you really need to use WGL is for a system palette change. Windows will indicate that such a change has been made by sending a WM_PALETTECHANGED message to every window, and then *OpenGLWindow::RedoPalette()* will take care of the change.

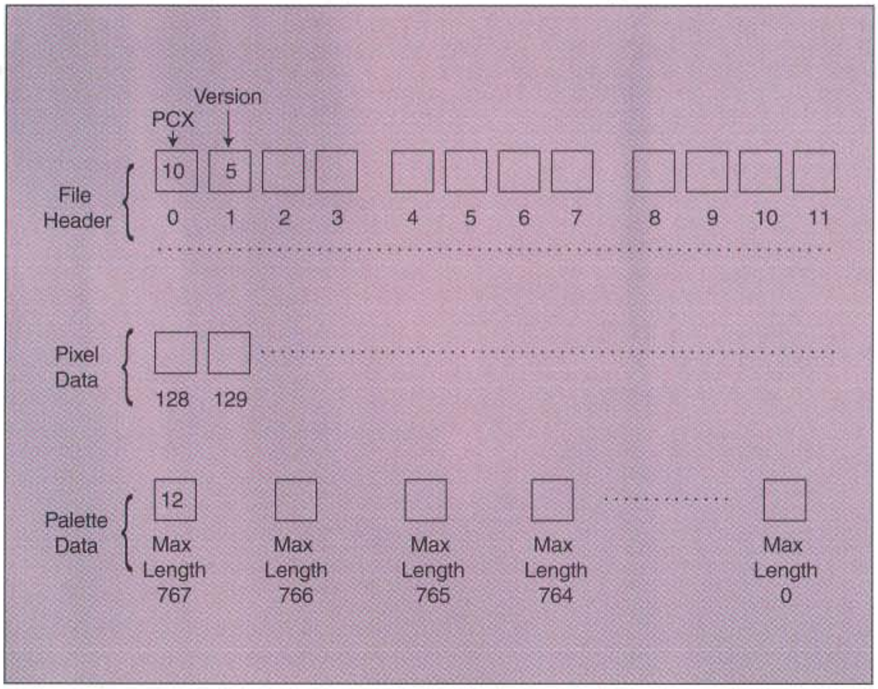


Figure 4: *The PCX format.*

Drawing the Entire Model

Inter.cpp's *redraw()* function (available electronically) redraws the entire model in its current state by specifying all of the triangle vertex coordinates and texture mapping coordinates between *glBegin(GL_TRIANGLES)* and *glEnd()*. This requires three calls to *glTexCoord2f()* (two parameters) and *glVertex3f()* (three parameters) for every triangle. One thing to note about the *glTextCoord2f()* function is that OpenGL expects texture-mapping coordinates to be relative, not absolute. To obtain these coordinate values, divide the original texture mapping coordinates from the model by their maximum range in the texture. In other words, divide the *S* component by the texture map's width and divide *T* by the texture map's height. These values will fall between 0 and 1 and remain unchanged when the texture is resized. For instance, (0.5, 0.5) will always point to the center pixel of the texture, no matter whether the texture dimensions are 173×233 or 256×256. Of course, doing a floating-point divide three times per loop is inefficient. By storing these values ahead of time the loop's efficiency could be improved greatly.

Between frame redraws the rendering window's window procedure keeps track of mouse movements and mouse button activity by listening to WM_MOUSEMOVE, and WM_*BUTTON(UP/DOWN) messages. The movement increments are then temporarily stored in two arrays— one for translational movements, and another one for rotations. At the beginning of each frame redraw the linear transformations are carried out using *glTranslate()* and *glRotate()*.

Code	Definition
GL_TEXTURE_2D	Defines a two-dimensional texture.
0	Supplies one texture as multiple resolutions.
4	Indicates which of the R, G, B, and A values are used.
scaledWidth	New width.
scaledHeight	New height.
0	Width of the border (no border).
GL_RGBA	Format of the texture data.
GL_UNSIGNED_BYTE	Data type of the texture data.
glTexture	Pointer to array containing texture to be rescaled.

Table 1: *Explanation of the parameters in* glTexImage2D(GL_TEXTURE_2D,0, 4,scaledWidth, scaledHeight,0,GL_RGBA,GL_UNSIGNED_BYTE, glTexture);.

Conclusion

Although OpenGL is straightforward to use, simply knowing the API is not sufficient. Since OpenGL does not provide functions to read 3D model and texture files of your preferred format, a basic understanding of 3D concepts and some amount of manual data manipulation is also required. Combining Win32 with OpenGL makes it possible to develop applications with user-friendly interfaces and impressive 3D graphics.

Keep in mind that one of OpenGL's bonuses is portability. Porting your Win32 OpenGL applications to X under UNIX should not be much more difficult than cutting and pasting some of the graphics code. Of course, creating another interface from scratch will be necessary.

DDJ

(Listings begin on page 96.)

The Kernel Graphics Interface

A portable high-performance graphics subsystem

Andreas Beck

Determining how an operating system should handle graphics is an exercise in tradeoffs. If you are interested in the fastest possible graphics performance, the only solution is for your application to work directly with the graphics hardware without regard to security. However, if you are willing to sacrifice a little bit of speed to gain portability and a degree of safety, GGI could help you a lot.

The GGI (General Graphics Interface) project (http://www.ggi-project.org/) is intended to bring safe, fast, and portable graphics to a variety of platforms and operating systems. GGI consists of user-level libraries of basic graphics functions and kernel-level drivers that handle the low-level graphics routines. The Kernel Graphics Interface (KGI) is the kernel console interface upon which the Linux implementation of GGI is based. Figure 1 shows how GGI and KGI are related. In this article, I describe the motivation, architecture, and implementation of KGI.

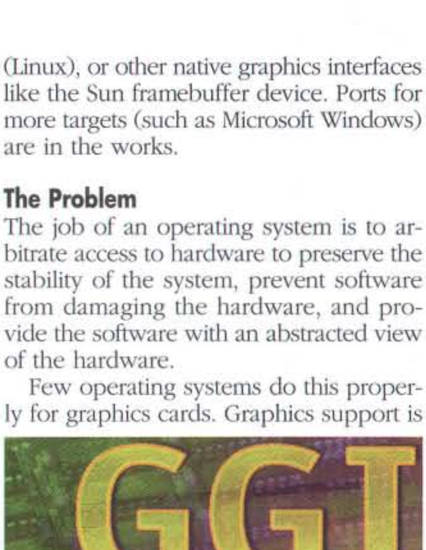
GGI is not confined to Linux, nor to KGI as the display subsystem. LibGGI is a lightweight graphics library that runs on a variety of platforms and graphics subsystems like X-Windows (tested on Solaris, AIX, IRIX, Linux, and others), SVGAlib (Linux), or other native graphics interfaces like the Sun framebuffer device. Ports for more targets (such as Microsoft Windows) are in the works.

Andreas studies physics at the University of Düsseldorf, Germany. He can be reached at andreas.beck@ggi-project.org.

The Problem

The job of an operating system is to arbitrate access to hardware to preserve the stability of the system, prevent software from damaging the hardware, and provide the software with an abstracted view of the hardware.

Few operating systems do this properly for graphics cards. Graphics support is either placed entirely in the kernel (like NT) or is left to user-mode applications with special permissions (like traditional Linux SVGAlib or X applications).



From a security point of view, there is nothing wrong with placing all graphics functionality in the kernel. The problem is that it vastly increases the kernel size at the expense of stability. Video drivers become more difficult to write and especially to debug— and errors in the drivers impact system stability.

On the other hand, the SUID root approach used by X and SVGAlib presents some security hazards. In general, you want to avoid running any applications as SUID root, since buggy or malicious code can easily be manipulated to break into, or simply break, a system.

A malicious, or merely carelessly programmed, graphics application can easily hang the system by causing a bus lockup (possible with many graphics cards due to bad programming), leaving the console in graphics mode (making it hard to use the system), or locking out virtual console switching. Worst of all, a malicious application might even be capable of damaging hardware by programming unsuitable clocks, thus overloading the RAMDAC and/or monitor. While most modern monitors have protection circuitry for this, RAMDACs are usually without defense.

X circumvents this problem somewhat by being a client-server system, which protects the privileged server from malicious or buggy user code. Yet even then, it is still possible to abuse the X server, for instance, to read any file on your system (see http://www.rootshell.com/).

SVGAlib is a bigger problem, because its applications must be SUID root. Consider the binary-only releases that are necessary for commercial games but must run SUID root. Would you trust all vendors not to spy on your system? Would you always check PGP signatures to make sure you don't have a hacked copy with some Trojan Horse? Even worse, normal users can't develop SVGAlib applications since root access is necessary to give appropriate permissions to the executable so it can be tested.

The Solution

KGI tries to address these problems by moving only the critical part— the actual programming of the graphics hardware— to the kernel. This reduces the security problems to those that any UNIX device

(continued from page 50)
exposes: inappropriate file system permissions and bugs in the driver.

KGI does not do the actual drawing in the kernel. It's not necessary, and doing so would increase the possibility of errors that are even more serious when they happen in a kernel context. The KGI driver is designed to be a thin layer around the hardware functionality. It only abstracts functions that are fairly standard between different cards.

Functions for setting up modes and some common accelerated drawing functions are available via a standard command API, while card-specific quirks are exported in a private command area that is called by a card-specific user-mode counterpart.

Implementation Considerations

Speed is the main problem with a graphics interface that is at least partially running in kernel mode. If you needed to make a kernel mode call every time you called a basic function like drawing a pixel, the system would crawl.

Fortunately, almost all available cards have some notion of a framebuffer, a portion of the onboard Video RAM (VRAM) mapped into the CPU's address space. Accessing the VRAM is normally considered a safe operation. Some hardware accelerator registers are mapped to VRAM, but these can normally be excluded by the kernel code via the MMU of the host CPU.

From user-mode, the KGI driver API exposes a command interface that needs to do a user-to-kernel transition (under Linux, an ioctl call to /dev/graphic), and a memory-mapped linear framebuffer, a continuous area in RAM that represents the VRAM contents.

Not every graphics card has a linear framebuffer. However, as those of you who are familiar with DJGPP may know, there is an elegant solution for this: the MMU. If the card exports a banked-style buffer (for example, a 64K window at 0xA0000, as old Trident 8900s did), it is mapped at the appropriate place in a virtual memory area as big as a linear buffer of the card would be. The other areas are marked to be swapped out. If such an area gets hit, the driver is notified, moves the card's window accordingly, and corrects the mapping.

There are some speed problems with this, because the MMU trap is expensive compared to just setting the bank with an "out" instruction. At the same time, due to the design of most such cards, we cannot export the banking register to user space anyway, because of security considerations (it is normally on an indirect register that also hosts CRTC timing, and so on). On the other hand, this approach leaves bank-crossing-detection to the MMU and thus saves unnecessary (sometimes nontrivial) checking code.

Now, we have a decent and fairly fast interface for all common tasks. All really primitive things that are not worth the overhead to call into the kernel (DrawPixel, very short lines, and so on) are performed via the MMAPed VRAM. More complex and administrative functions are performed via the command interface.

One other catch is that you probably do not want to write any emulation code into the drivers for cards that do not have a particular function accelerated. Microsoft's DirectX handles this problem using capability bitmaps. Having capability bitmaps means that you can query to see if an acceleration function is available via some kind of a bitmap or test for a NULL pointer. In our opinion, this is too hard to extend, because you have to extend the bitmap or table with every new version, making lots of revision checks necessary to see if a particular capability is accessible in a given revision at all. So we chose another way to handle software fallback for our acceleration code.

An accelerated function call always returns a status code that either says "completed successfully" or an error code that suggests what to do instead and also how long that information is valid.

The suggestion can say:

- CANNOT: This is returned for hardware-specific operations or context-sensitive operations (for instance, trying to set the frame for video overlay on a board without such capability).
- USE_LOWER: This is used when it is most likely a good idea to use a set of simpler acceleration calls (for example, using multiple horizontal lines to draw a box), because the resulting calls would be accelerated.
- USE_MMAP: This is returned when no simpler accelerator calls are supported. Thus, it is advisable not even to try them, but rather to directly draw on the MMAPed VRAM.

The expire information tells how long this information is valid. This allows us to avoid having to call the accelerator function each time for cases where a certain accelerator function may be only temporarily unavailable.

- NOW: Retry next time. It can't be done just now, because the accelerator is too busy or some similar problem that is likely to go away the next time the function is called.
- GC: Retry when the graphics context has changed (for example, if the accelerator cannot draw with a given raster operation).
- MODE: Retry when the mode has changed (that is, if the accelerator cannot be enabled in a specific mode as in the VGA compatibility modes of many common accelerators).
- ALWAYS: The accelerator never has this capability.

The advantage of handling software fallback this way over a DirectX-style bitfield is that this is extensible in a compatible way on both kernel and user sides. A newer KGI driver will know some new command codes that older libraries won't know about. So, you could lose a bit of extra acceleration with older libraries, but it's better than being incompatible.

A newer library may use some command codes that are not supported by older drivers. This triggers a "default" case that deals with the commands and always returns ENOSUP_ALWAYS_LOWER or ENOSUP_ALWAYS_MMAP (depending on whether or not the driver has a reasonable base set of accelerated commands). This return code causes the library to permanently disable the accelerator call after the first try and use an emulation routine instead. Again, you may lose a bit of potential acceleration if your kernel isn't up to date with the library, but it still works.

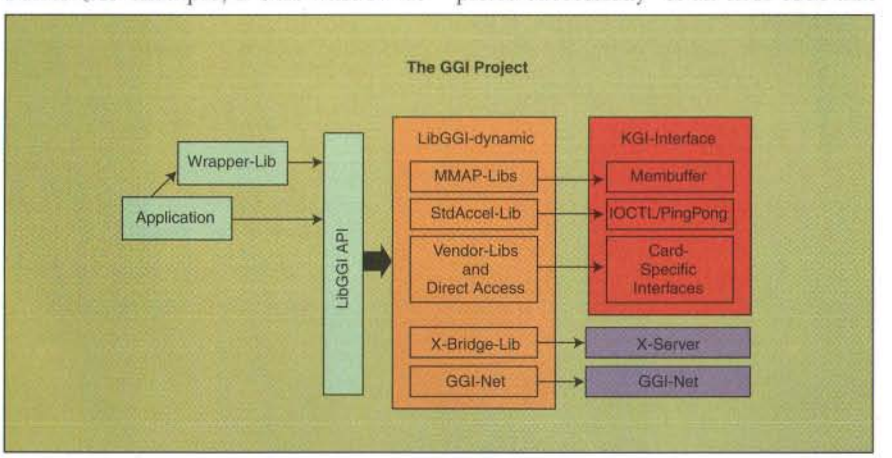


***Figure 1:** How GGI and KGI are related.*

IT'S A CODE PERFORMANCE PROBLEM.

BUT WHERE?

ONLY NuMEGA TrueTime™ PROVIDES SEAMLESS MULTI-LANGUAGE PERFORMANCE ANALYSIS FOR COMPONENT-BASED APPLICATIONS.

Now, for the first time, you can easily and accurately pinpoint code performance bottlenecks no matter what language they are written in or where they reside. In Visual Basic, Visual C++, and Java, NuMega TrueTime lets you see exactly where problems occur. Get NuMega TrueTime and get all your applications and components up to speed.

Microsoft Enterprise Development Partner

Compuware® NuMEGA™

Call to request a **FREE 14 DAY LICENSE.**

1-888-686-3430 www.numega.com

SmartDebugging™ for faster development VISUAL BASIC® VISUAL C++® JAVA™

30-DAY MONEY BACK GUARANTEE

(continued from page 52)

Enhancements

While the scheme described earlier is enough for normal applications, there have always been some drawbacks to this approach:

- Only relatively common acceleration commands are supported. Adding all the card specifics would result in an incredible number of commands. In addition to the number of commands increasing astronomically, it is very possible that different accelerator functions in two different drivers could end up using the same command codes, as drivers are developed independently.
- There is no direct way to get at the acceleration registers, even if this is otherwise safe to do (which it is for a few very high-end cards).
- Some cards have multiple memory areas for textures, overlays, and so on.
- There is no way to support display lists or similar things that would dramatically reduce the number of user-to-kernel transitions and, thus, overhead.

To overcome these limitations, KGI allows exporting additional API functions that allow you to circumvent these problems:

- Private commands. KGI reserves an area for private command codes. These are handled by a card-specific library in user space to make the best possible use of the card.
- Mapping of card Memory-Mapped IO (MMIO) areas, or eventually allowing access to the card's ports if this is safe (up to now, we have not found cards where port access is safe). Here, too, card-specific libraries are used to convert the card-specific API represented by the MMIO area to the common API.
- Mapping of cards' additional memory areas like texture memory, YUV overlay planes, and so on.
- PingPong buffers, which are simply filled with commands (all in user space) and then executed with a single command (one user-to-kernel transition). This operation can be done asynchronously with the program continuing to execute on the host CPU, while the accelerator is fed with commands using either DMA, accelerator-generated "accel-idle" or "accel-buffer-lowwater" interrupts, or host-generated timer interrupts. This allows for maximum throughput, as the host CPU can prepare the next drawing commands while the accelerator is still drawing the last batch.

Multiple APIs and Libraries

I have talked about having multiple APIs. How do you know which particular APIs are present and how to make use of them? How do you avoid a horrible mess where the applications must know all of the APIs?

This is one of the reasons for LibGGI, which consists of a basic stub library and a rather large bunch of API libraries that build the bridge between the various hardware (or software—LibGGI can also be used to display in an X-Window) APIs and the LibGGI API. When setting up a mode, LibGGI asks the target (KGI in our case) for a list of the exported APIs, a set of strings that classify how you can access various card features. Figure 2 shows a typical API list. The meanings of the strings, which are listed in increasing order of precedence; see Table 1.

```
"generic-linear-8"
"generic-ioctl"
"generic-ramdac"
"S3-generic"
"S3-virge"
```

Figure 2: *Typical API list.*

The libraries are loaded in a way that allows more specific functions to overload the more generic ones, automatically yielding a startup configuration that always uses the best available function. In some cases (as with the ioctl API), these entries can be altered at run time if functions are not available.

One problem remains. LibGGI can only make use of functions that are needed for implementing the LibGGI API. If you look at these functions, you will realize that they account for few of the functions a card can support.

We have decided to keep LibGGI small to save space for simple applications and things like embedded systems. For more complex functions, LibGGI allows the registration of extensions like LibGGI2d and Mesa-GGI, which add support for the APIs necessary for specific tasks.

Implementation Details

Additional goals with the design of KGI included:

- Easy driver writing.
- Modular design for cards that are made from similar components (S3 cards with different RAMDACs, clocks, and so on are a good example).
- A simple way to enhance drivers for fairly compatible future generations of known cards.
- Full abstraction from the operating system for easy portability.

These are achieved by using a modular design approach that makes every KGI driver consist of six basic modules:

- Chipset module. This controls all functions related to mode setup, CRTC programming, RAM timing characteristics, interfacing RAMDAC and Clock, and so on.
- Clock module. This controls the pixel clock generation. This is separated from the chipset driver, as there are cards (S3, for instance) that have the clock as a physically distinct chip, with the different cards made by combining basic chipset, clock, and RAMDAC chip in different ways.
- RAMDAC module. The RAMDAC modules is similar to the clock module, but controls the RAMDAC features like palette setting, VRAM-bus activation, RAMDAC-internal hardware cursors, Gamma correction, and the like.
- Graphics (accelerator) module. Some chips have the acceleration engine either detached from main chipset or use the same or very similar acceleration engine on different chipset versions. Thus, separating acceleration programming

The EvStack Kernel Enhancement

EvStack is an extremely flexible console system we designed to overcome some limitations of the current Linux-KGI kernel patch, which breaks some features and programs (notably XFree and SVGAlib). The basic idea behind EvStack is to pass events between independent modules instead of hardwiring the calls. This allows you to plug together a console and dynamically swap out parts, like the VT-emulation. Under EvStack, you can have xterm, Linux, and dumb consoles on the same machine as well as different fonts, screen sizes, and screen modes (for instance, graphical consoles) on the different virtual terminals. With the EvStack patch installed, you can do one of three things:

- Turn EvStack off at compile time, giving you traditional Linux console code.
- Turn EvStack on, but load or compile in the conlinux.o module, giving you traditional Linux behavior running on the new code.
- Turn EvStack on, but don't load conlinux.o, giving you pure EvStack behavior, with the additional conlinux API disabled and all configuration occurring via */proc*.

—A.B.

from the other aspects of the card makes sense (that is, all newer S3 cards can be run with the S3 generic acceleration driver). Not all of the capabilities of very new cards would be used, but driver development is eased quite a bit, since you can try out your new chipset driver without having to write a graphic module.

- Monitor module. What features are there in a monitor that will need a driver? At the very least, such things as timing limitations, ensuring that the image is centered on the screen, power-saving capabilities, and more. Being able to use any of these requires some knowledge about what the monitor supports. The monitor driver allows safe access to these features, and automatically chooses suitable timings.
- Kernel module. This does the interfacing to the host OS. It implements access methods to the hardware, to PCI services, and so on. In theory, we should be able to run the same KGI driver on different operating systems by just linking with a different kernel module. (We have not yet tried this because we are currently restructuring the Linux console. Porting efforts now would result in a lot of duplicate work.)

Conclusion

What does Linux gain by using KGI? First, the graphics card is handled like any other device, which means that arbitration and access to critical registers occur in one central place—the kernel.

Second, since the kernel is able to control the graphics card, we have a few new capabilities:

- A real Secure Attention Key (SAK) that can kill off graphical applications safely because the kernel itself is able to reset the graphics card to a sane state.
- Simple and safe resizing capabilities for VTs. For example, with KGI, you can implement VT100 ESC codes that were impossible to implement without these resizing capabilities.
- Support for graphical consoles, thanks to the new EvStack kernel enhancement (see the accompanying text box entitled "The EvStack Kernel Enhancement"). This is immensely desirable for hardware that has no VGA-like text mode or for languages that require the ability to represent more than 256 characters.
- The ability to operate the graphics card in MMIO mode, which means that the registers of the card are mapped to a programmable place somewhere in the memory address space, thereby freeing the VGA registers in IO space. As a result, Linux/KGI is multihead capable with cards that support that feature.

String	Meaning
S3-virge	This is an S3 Virge card. If you have a specific library that knows the API and the Virge-specific functions, then load it.
S3-generic	The kernel knows which functions are available on all S3 cards.
generic-ramdac, generic-ioctl	The generic RAMDAC APIs are supported, as is the KGI-ioctl interface.
generic-linear-8	The card (for this mode, anyway) has a linear framebuffer with eight bits per pixel.

Table 1: *The meaning of the strings in Figure 2 (from bottom to top).*

Third, together with LibGGI, you have a lightweight, portable, and fast graphics subsystem. (A single-disk demo that uses a mere 700-KB compressed image is available electronically; see "Resource Center," page 3, or my home page at http://sunserver1.rz.uni-duesseldorf.de/~becka/.) This is of special interest for embedded systems, which can now use Linux instead of relatively expensive and less open ("nice README, but where is the source?") solutions like QNX or Windows CE.

Finally, you will no longer have dangerous SUID root graphics applications. The GGI project has developed both a wrapper library that allows most SVGAlib applications to run without root permissions, and a replacement X server called Xggi.

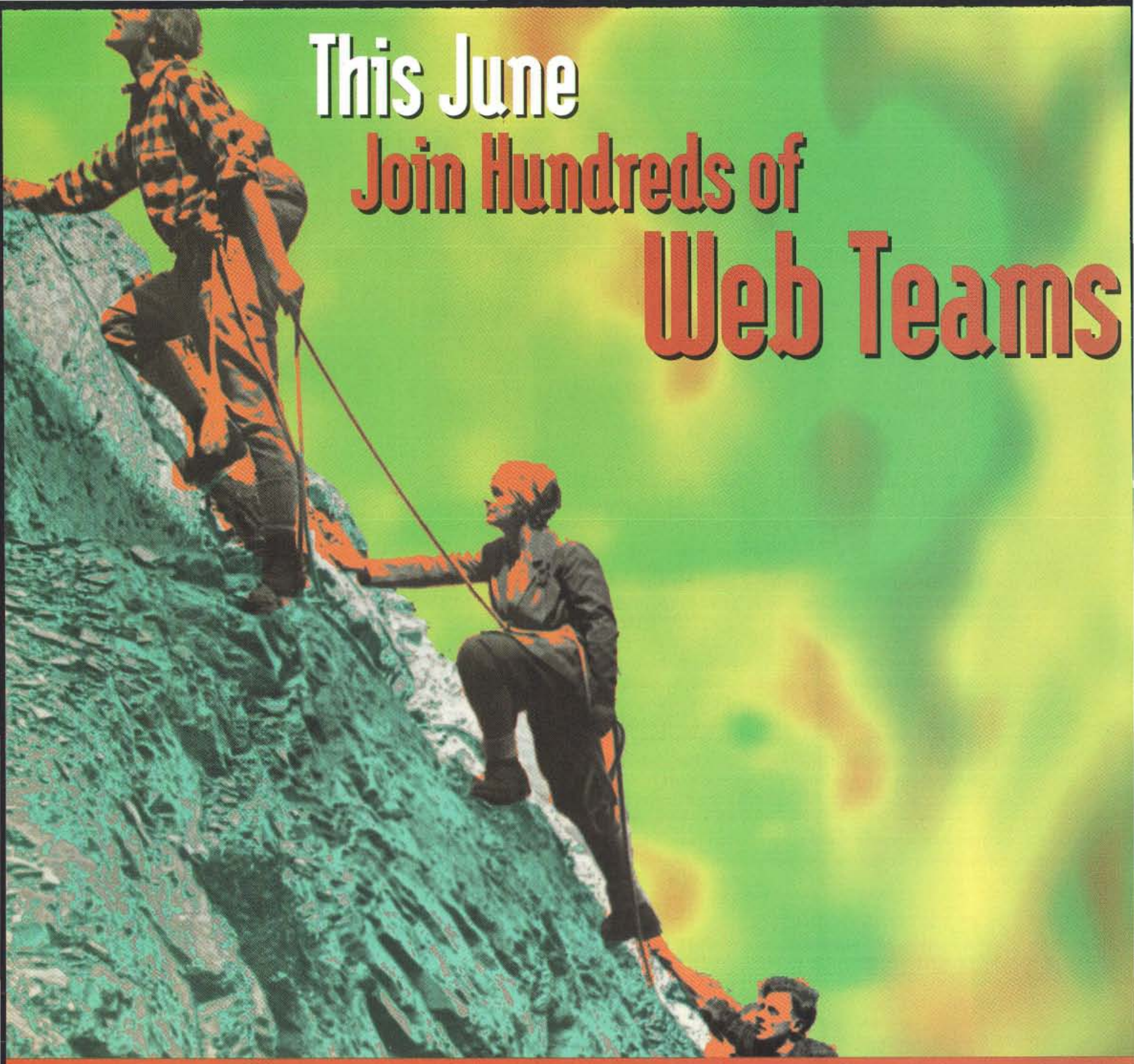
Resources

The GGI homepage (http://www.ggi-project.org/) contains snapshots of the latest source, instructions on how to obtain them via CVS, links to GGI-relevant web sites, and up-to-date information about the project. Our mailing list is hosted at ggi-develop@eskimo.com. Subscription information is found on the GGI web site. If you plan on subscribing, be prepared—the list has high traffic.

Acknowledgments

Thanks to the GGI development team, especially Steffen Seeger, Jason McMullan, Emmanuel Marty, Ben Kosse, and Michael Krause for their work and for reviewing this article and correcting several glitches. I'd also like to thank S3, Cyrix, 3Dlabs, the FLUG for providing the GGI development team with information and donations, and all the users and testers of GGI.

DDJ


This June
Join Hundreds of
Web Teams

Affine Texture Mapping

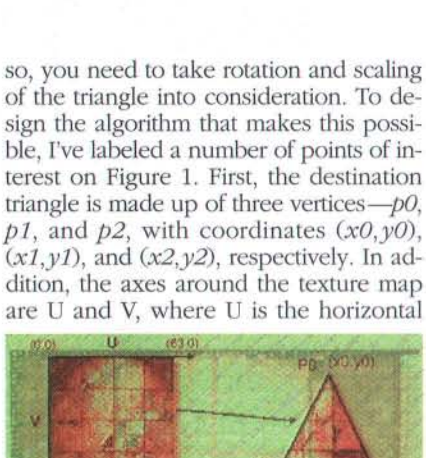
A fundamental technique for graphics programmers

André LaMothe

It used to be you could get away with developing flat shaded 3D computer games and engines. But if your software doesn't support texture mapping these days, it will likely end up in the bargain bin. One form of texture mapping is "affine" texture mapping, which is fundamental to many forms of 3D rendering, including light interpolation and other sampling type operations. In this article, I'll present an affine texture mapper that can texture map a 64×64-pixel/256-color rectangular bitmap onto a triangular polygon with full texture coordinate support. In addition, I'll include a demo that loads texture maps and draws thousands of textured triangles a second. Although this demo is in DirectX, the ideas and concepts are applicable to other systems. And since the texture mapper is in straight C, it's totally portable.

Getting Down to Specifics

Assume you want to texture map a rectangular bitmap that is 64×64 pixels in 256 colors (one byte per pixel) onto an arbitrary triangle with any coordinates. To do so, you need to take rotation and scaling of the triangle into consideration. To design the algorithm that makes this possible, I've labeled a number of points of interest on Figure 1. First, the destination triangle is made up of three vertices—*p0*, *p1*, and *p2*, with coordinates *(x0,y0)*, *(x1,y1)*, and *(x2,y2)*, respectively. In addition, the axes around the texture map are U and V, where U is the horizontal axis and V is the vertical axis. Both U and V range from (0,0) in the upper left to (63,63) in the lower right. What you want to do is design an algorithm that samples the texture map, so that the sampled pixels can be used to color each pixel of each scanline of the target triangle polygon as it is being rendered.



There are a number of ways to draw triangles, including tracing the edges of the triangle with a line-drawing algorithm (such as Bresenham's) or with simple interpolation. I prefer interpolation since it's more straightforward. Also, the concept of interpolation is important because the texture mapping algorithm is based on it. In Figure 2, all you have to do is find the points (shown as little dots) that make up the integer rasterized version of the triangle. Once you find these dots for each scanline that makes up the triangle, drawing the triangle is nothing more than performing a memory fill from dot to dot. Finding these points simply involves interpolating the slope of each side of the triangle. The interpolation is done as follows:

You know that the height of the triangle is:

```
dy=(y2-y0);
```

and the difference in the "x" between the lower-left vertex and the lower-right vertex is:

```
dx_left_side=(x2-x0);
dx_right_side=(x1-x0);
```

Thus, the slope of the left side is:

```
slope_left_side=dy/dx_left_side
    =(y2-y0)/(x2-x0);
```

And, the slope of the right side is:

```
slope_right_side=dy/dx_right_
    side=(y2-y0)/(x1-x0);
```

However, you don't exactly want the slope. The slope is the "change in Y per change in X." This means that if you were to move over exactly one pixel in the X direction, then the Y would change by the slope. You don't want this. In fact, you want the opposite—*dx/dy*—because you are drawing the triangle scan line by scan line and incrementing Y each time; hence *dy=1*, which is a constant. Thus:

```
dx_left_side=1*(x2-x0)/(y2-y0);
```

and

```
dx_right_side=1*(x1-x0)/(y2-y0);
```

Listing One (listings begin on page 96) is a pseudocode implementation of the triangle drawing algorithm.

André is the author of Tricks of the Game Programming Gurus, Teach Yourself Game Programming in 21 Days, The Black Art of 3D Game Programming, *and his latest creation,* Windows Game Programming for Dummies. *He can be contacted at necron@slip.net or at http://xgames3d.com.*

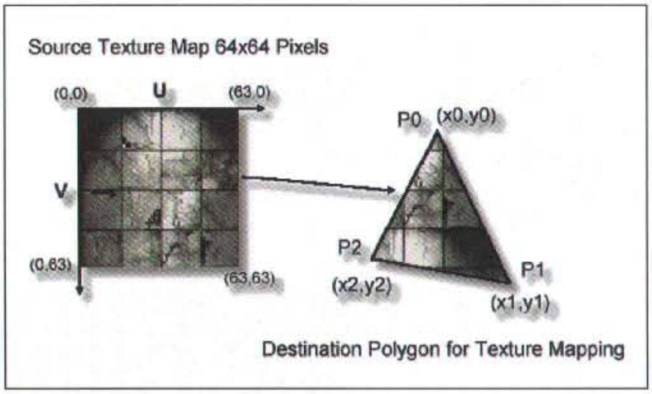


Figure 1: *Texture mapping source-to-destination labeling.*

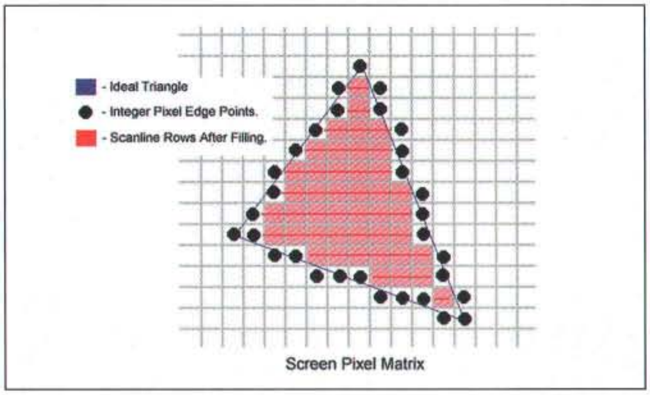


Figure 2: *Screen pixel matrix.*

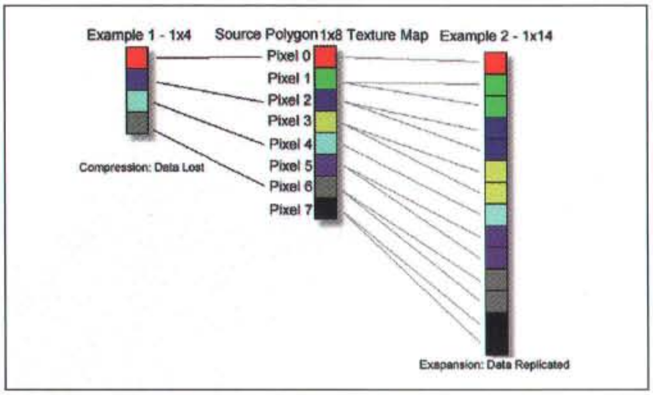


Figure 3: *1D texture mapping: (a) 1×4; (b) 1×14.*

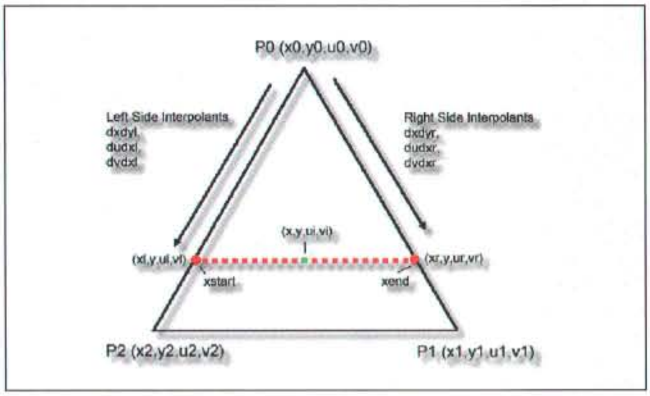


Figure 4: *Graphic representation of texture-mapping algorithm.*

One Dimensional Interpolations

Texture mapping a triangle with a rectangular texture map involves lots of interpolating. Consequently, it's easy to make a mistake or to write a slow algorithm. With this in mind, I'll start with the simplest case in one dimension. Figure 3 illustrates the simplest texture mapper—the texture mapping of a single vertical line that's one pixel thick and eight pixels high.

What you need to do is "sample" the texture map (in this case, a single 1×8 pixel bitmap) and map it into the destination polygon, which is 1×*n* pixels, where *n* can range from one to infinity.

As a first example, assume that your destination polygon is 1×4 pixels. It makes sense that you want to sample the source texture every other pixel, as in Figure 3. Thus, if you select pixels (0,2,4,6) of the source texture and map them into the destination polygon at positions (0,1,2,3), then you are doing pretty good. But how did you arrive at (0,2,4,6)? The answer is by using a sampling ratio, which is nothing more than an interpolation factor. In general, *sampling_ratio=source_height/destination_height*. Thus, the sampling ratio is *sampling_ratio*=8/4=2. Thus, every one pixel you move on the destination polygon in the vertical axis, you must move two pixels on the source to keep up. That's where the "two" comes from and hence the sampling sequence (0,2,4,6). Unfortunately, this means you had to throw away half the pixels. This is a problem with sampling on an integer matrix without any averaging. If you were writing a high-end 3D modeler (like 3D Studio MAX), then you would probably average the pixels you're sampling (area sampling) to get a better approximation, but for games and real time, our technique will do.

In the previous example, the source texture was compressed; that is, the destination was smaller than the source and information was lost. On the other hand, there could be the case that the destination is bigger than the source, and there isn't enough information to go around. In this case, the source data must be sampled more than once and replicated. This is where all "chunkiness" comes from when texture mapped polygons get too close to you in a 3D game. There isn't enough texture data so some sample points are sampled many times, creating big blocks. Referring again to the second example in Figure 3, you see that the source is again 1×8, but this time the destination is 1×14 pixels. Obviously, you need a fractional sampling ratio. Again, *sampling_ratio=source_height/destination_height;*. Thus, the sampling ratio is *sampling_ratio=8/14=0.57.*

Hence, the sample for every pixel you draw on the destination polygon should be taken 0.57 units from the last sample point on the source. This gives you the following sample point sequence for destination pixels (0,1,2,3,....13):

```
Sample 0: 0.57
Sample 1: 1.14
Sample 2: 1.71
Sample 3: 2.28
Sample 4: 2.85
Sample 5: 3.42
Sample 6: 3.99
Sample 7: 4.56
Sample 8: 5.13
Sample 9: 5.7
Sample 10: 6.27
Sample 11: 6.84
Sample 12: 7.41
Sample 13: 7.98
```

To get the actual sample points, you simply truncate the sample points in integer space or take the floor of each value resulting in the sample points (0,1,1,2,2,3,3,4,5,5,6,6,7,7), which sounds about right. Each point got sampled about two times, or 1/0.57.

Multiple Interpolations

When I wrote my first affine texture mapper, I thought something must be wrong since it seemed like I was interpolating everything. The truth is, there is really no way around all the various interpolants, and in the end, the inner loop

for each pixel can be optimized into around 10 cycles/pixel on a Pentium, which translates to a theoretical maximum of 10- to 20-million textels (textured pixels) per second on a 100-MHz Pentium.

The idea behind the algorithm is that you want to interpolate down the left and right edges of the triangle and draw each scanline strip as we go with the proper texture pixels. What you need to do first is assign full texture coordinates to the vertexes of the destination triangle to give us a frame of reference for the interpolants. Thus you must assign each vertex a (u,v) texture coordinate, as in Figure 4. Therefore, each vertex has a total of four data components—that is, it's a 4D value. Since the source texture map is 64×64 pixels, the texture coordinates must range from 0–63 for any vertex. This will map or stretch the texture map to each vertex.

Figure 5(a), for example, has the texture coordinates (0,0), (63,0), and (63,63) mapped to vertices 0,1, and 2, respectively. This basically copies half of the texture map to the destination triangle, which is what you would expect. In Figure 5(b), you see the same texture mapped onto two triangles which are adjacent to each other forming a square. In this case, the texture coordinates are selected in such a way that half of the texture map is mapped to one triangle and the rest to the other, hence, a perfect texture wrapping around two triangles. Moreover, this is how you would make a quadrilateral; that is, with two triangles. Now that you have a visual on the problem and know the labeling from Figure 4, let's implement the algorithm mathematically. The variable names used in the following analysis are based on Figure 4 and the final program so that you can follow the program code more easily.

The left edge interpolants are:

```
dxdyl   =   (x2-xØ)/(y2-yØ);
  // x interpolant for left side
dudyl   =   (u2-uØ)/(y2-yØ);
  // u interpolant for left side
```

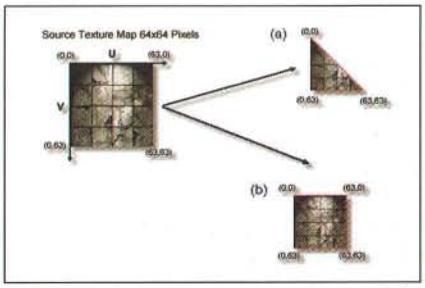


Figure 5: *Texture mapping 3- and 4-sided polygons. (a) single triangle; (b) two triangles making a quadrilateral.*

```
dvdyl   =    (v2-v0)/(y2-y0);
  // v interpolant for left side
```

Similarly, the right edge interpolants are:

```
dxdyr   =    (x1-x0)/(y2-y0);
  // x interpolant for right side
dudyr   =    (u1-u0)/(y2-y0);
  // u interpolant for right side
dvdyr   =    (v1-v0)/(y2-y0);
  // v interpolant for right side
```

There's a lot of room for optimization. For example, $(y2-y0)$ is common and need only be computed once. Furthermore, it's better to compute the reciprocal of $(y2-y0)$ and then multiply.

The interpolants must be in reference to some starting point. This starting is the top-most vertex, vertex 0. Hence, you need to start the algorithm off in the following manner:

```
xl = x0; // starting point for
  left side edge x interpolation
ul = u0; // starting point for
  left side edge u interpolation
vl = v0; // starting point for
  left side edge v interpolation
```

And for the right side,

```
xr = x0; // starting point for
  right side edge x interpolation
ur = u0; // starting point for
  right side edge u interpolation
vr = v0; // starting point for
  right side edge v interpolation
```

Now you can interpolate down the left and right edges with:

```
xl+=dxdyl;
ul+=dudyl;
vl+=dvdyl;
```

and

```
xr+=dxdyr;
ur+=dudyr;
vr+=dvdyr;
```

At each point on the left and right edge of the triangle, you still need to perform one more linear interpolation across the scanline. This is the final interpolation and the one that will give you the texture coordinates *(ui,vi)*, which you'll use as *[row, column]* indexes into the texture bitmap to obtain the textel. All you need to do is compute the *u,v* coordinate on the left and right side, then use the *dx* to compute a linear interpolation factor for each. Here's the math:

```
dx = (xend-xstart);
  // difference or delta dx
xstart = xl;
  // left starting point
xend = xr;
  // right starting point
```

Therefore, the interpolants across each scanline in *u,v* space are:

```
du = (ul-ur)/dx;
dv = (vl-vr)/dx;
```

Then with *du,dv*, you have everything you need to interpolate across the scanline at vertical position y from *xstart* to *xend*; see Listing Two.

Conclusion

That's it. Of course for the outer loop, you would still interpolate *xl,ul,vl,xr,ur,vr* down the triangle edges for each scanline of the triangle.

The files tmapper.h and tmapper.cpp (available electronically; see "Resource Center," page 3) provide a complete implementation of the texture mapper. The program assumes a specific input data structure and that the texture map is a linear bitmap 64×64 pixels. Other than that, it's nothing more than an implementation of the derivation here, along with all the triangle cases and clipping. In addition, the program tmapdemo.cpp (available electronically) is a complete DirectX demo of the texture mapper that draws random triangles all over the screen in 640×480×256. Finally, BOX2.EXE is a 3D demo written by Jarrod Davis that uses the texture mapper.

DDJ

(Listings begin on page 96.)

Inside DVD

More storage, more features, same size

Linden deCarmo

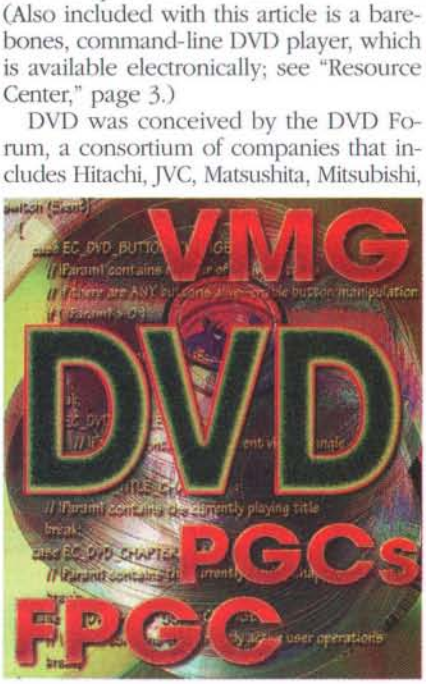
Although DVDs physically resemble CD-ROMs (five inches in diameter and 1.2 mm in thickness), DVD stores between seven and 25 times more data. This huge storage capacity makes it an ideal distribution vehicle for full-length movies (up to four hours long), high-quality audio (the contents of up to 13 CDs can be stored on one dual-layer DVD), and similar applications (not to mention data storage). DVD has garnered support from all major electronics and computer companies, and many major movie and music studios.

Even though DVD technology is promising, technical details about it are scarce— in part because those details are still being worked out. Currently, specifications have been agreed upon for DVD-Video and DVD-ROM. As its name suggests, DVD-Video is for video programs and is played in DVD players connected to TVs. DVD-ROM, on the other hand, stores computer data and is read by DVD-ROM drives connected to computers. Variations on DVD-ROM include those that are recordable one time (DVD-R) or many times (DVD-RAM). Most computers with DVD-ROM drives can also play DVD-Videos. Finally, there's the DVD-Audio format, for which technical specs haven't yet been finalized.

Linden is a software engineer at Oak Technology where he is currently working on the Interactive DVD Browser, the first publicly available DirectShow DVD environment. You can contact him at lindend@ ibm.net.

In this article, I'll examine how a DVD-Video (or simply DVD) player operates, examine the features of a DVD title, and investigate the interactive capabilities in both computer and consumer DVD titles. (Also included with this article is a barebones, command-line DVD player, which is available electronically; see "Resource Center," page 3.)

DVD was conceived by the DVD Forum, a consortium of companies that includes Hitachi, JVC, Matsushita, Mitsubishi,



Philips, Pioneer, Sony, Thomson, Time Warner, and Toshiba. Although no one "owns" DVD, companies making DVD products must license patented technology from a pool of companies.

One result of this collaboration is the multivolume series *DVD 1.0 Specification for Read-Only Disc* (ordering instructions are available at http://www.mpeg.org/MPEG/DVD/General/Order.html). The most interesting book in this series is *Volume Three*, which focuses on DVD-Video— a combination of a reference player design, optical media format, and multimedia data structures. The DVD-Video specification describes the required features to which a hardware-independent virtual machine must adhere. It also defines the assembly-language opcodes that have to be interpreted, the state diagrams the player must enforce, the system registers that can be manipulated, and the size and capabilities of user-accessible memory.

The DVD-Video Specification

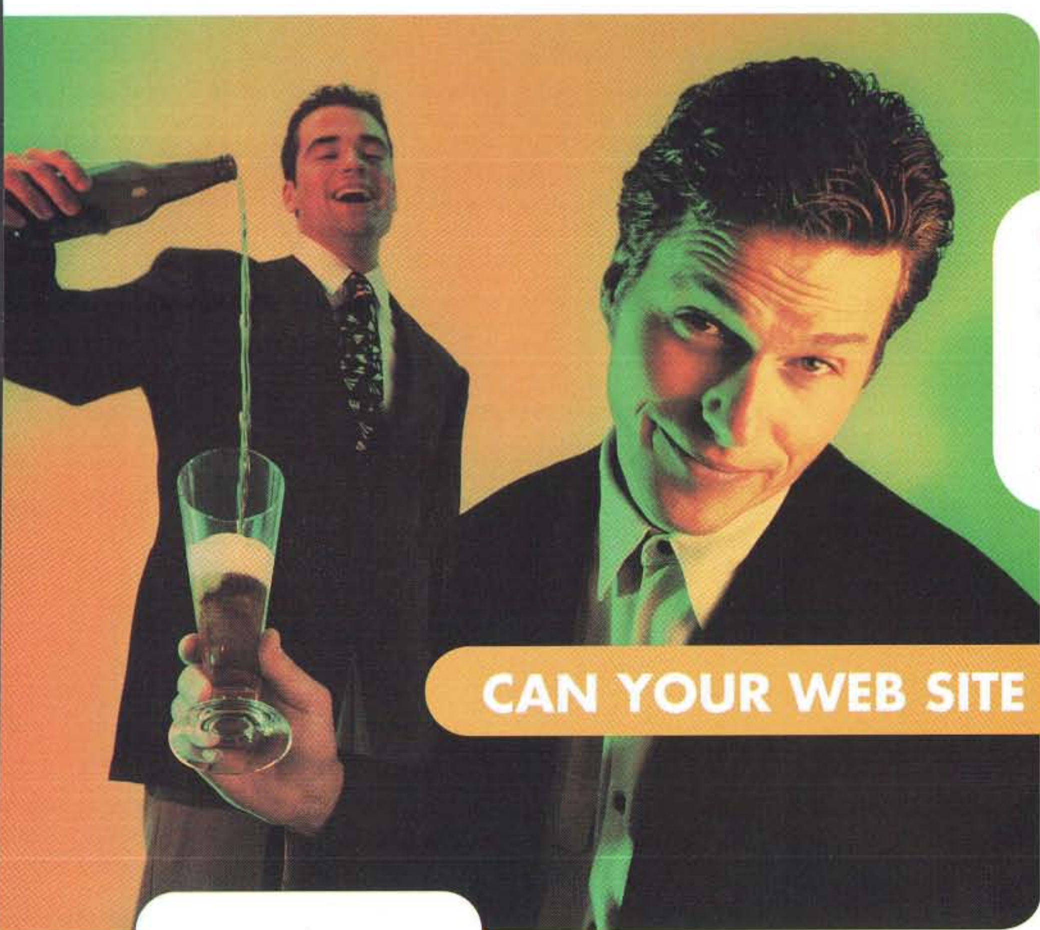
The DVD-Video virtual machine contains a low-level, assembly-like, instruction set with the usual branch, compare, and set operations found in most processors. However, the specification also has unique opcodes specifically designed for interactive presentations. For example, there are instructions to monitor parental controls, jump to specific locations in a presentation, and dynamically switch audio and video tracks.

All DVD players have at least 20 system parameters (or registers) that can be accessed only by privileged opcodes. For instance, there are instructions to change the currently playing audio stream and update the system register, which monitors the currently playing audio stream number. The player also offers 16 general-purpose parameters that you can modify without special instructions.

Every DVD-Video disc contains a *video_ts* (or video title set) directory, which consists of files with IFO or VOB extensions. VOB files store multimedia data, whereas IFO files instruct the player how to play the content in VOB files. There are two types of IFO files— Video Manager (VMG) and Video Titles Set (VTS).

DVD Video Manager

VMG is found in the video_ts.ifo file and it is the first file all DVD players read. This file is similar to a boot sector on a floppy disk— it supplies the player with initialization information and then points



(continued from page 64)
the player to where it can obtain the remainder of the data necessary to continue playback.

VMG contains vital information such as the name of the disc, number of titles on the disc, and optional information such as parental controls (parental controls prevents little junior from looking at violent or explicit material). Although DVD defines eight numerical levels of parental protection, the name associated with each level may differ between countries (for instance, the Canadian rating system differs from the United States system). In all cases, higher parental level values always permit more content to be viewed (see the accompanying text box "Parental Levels"). For instance, Parental Level Seven ("NC-17" in the United States) enables you to see more movies than Level One ("G" in the U.S.).

A VMG may also contain a feature known as the "VMG Menu" (VMGM), which gives users an overview of the disc's contents and potentially lets users jump to specific points in the title. It is composed of a video stream and an optional audio stream and a subpicture stream. To avoid bugs in the first generation of players, most early menus used MPEG-2 still images and had minimal interactivity. Because newer players are more stable, innovative authors are including full-motion video and surround sound in their menus.

Interactive features in menus (such as background audio and video) are displayed and controlled by data structures called "Program Chains" (PGCs)— arrays of programs, each of which normally represents a screen within a menu or chapter in a movie title. Each program contains one or more cells. Cells let you divide menus or chapters into more granular or logical subdivisions. They last a finite period of time, may have command instruction (or DVD assembly opcode) associated with them, and can enforce a delay when they complete playback. Although few titles take advantage of multiple cells per program, the DVD specification enables this feature to support effects such as slide shows in which playback must pause for a specific period of time after displaying a cell (or image).

Besides programs, PGCs may contain up to 256 navigational commands (128 of which may be executed before the programs in the PGC are presented, and 128 thereafter). These navigational commands are used for interactive purposes such as modifying the current video angle.

Every PGC also contains User OPerations (UOPs), which are stored in 32-bit fields where each bit (or individual UOP) represents the status of a unique interactive function on the player. Because these UOPs dictate which features in a PGC are legal (or usable), they have been almost as controversial as the region codes (see the accompanying text box entitled "Region Management"). To illustrate why this feature is so contentious, examine the Fast Forward UOP bit. If this bit is set, the DVD player cannot fast forward for the duration of that PGC. As a result, tricky content creators can embed commercials in DVD content and users will not be able to fast forward past them!

Attached to the tail of the VMG is the First Play PGC (FPGC). Once the DVD player is initialized, it searches for this PGC and executes the navigational commands inside of it. Most titles contain FPGCs that cause the player to display the VMGM, although it is possible for the

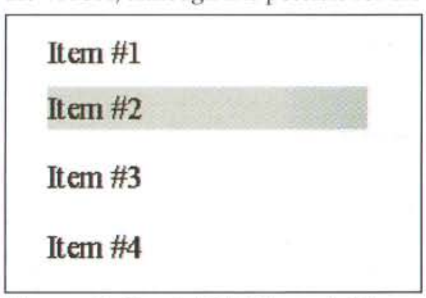


Figure 1: *How highlights work. The DVD player manipulates the color and contrast of a rectangle within the subpicture and this causes the area to appear highlighted. Here, the contrast for Item #2 is emphasized so that it appears selected.*

(continued from page 66)
FPGC to bypass the menu and jump directly into a movie scene (this is the technique that movies such as *The Mask* use to initiate immediate playback of a title).

Besides the video_ts.ifo file, the VMG also contains a Video OBject (VOB) file named video_ts.vob. VOB files are divided into packs each of which may contain a different media stream (packs are similar to a chunk in a WAV file). Although a pack may contain any data type, the DVD specification has stringent definition for video, audio, and subpicture packs.

Video packs in a VOB file normally contain MPEG-2 video. Although the MPEG-2 video format was defined by a standards body, it supports myriads of options that make it difficult to create a robust decoder. Therefore, to enhance compatibility and reliability, DVD places the restrictions of limited choice of resolution and maximum bitrate guidelines on MPEG-2 video content.

For NTSC locales (North America and Japan), the MPEG-2 video stream resolution in DVD must be 720×480, 704×480, 352×480, or 352×240. PAL (or European) resolutions must be 720×576, 704×576, 352×576, or 352×288. Furthermore, whatever video resolution and audio compression routines are used, the content cannot exceed a sustained bit-rate greater than 10.08 Mbits/sec.

The designers of DVD also delineated what audio packs may appear in a DVD stream. The audio types supported in the initial DVD specification include: Pulse Code Modulation (PCM), Dolby Digital (AC-3), MPEG-2 audio, Digital Theater Sound (DTS), and Sony Dynamic Digital Sound (SDDS). PCM is commonly used in stereo sound tracks and is identical to PCM content found in Windows, UNIX, and Macintosh (although DVD supports higher PCM resolutions and sampling rates than these environments).

If the content contains multichannel sound, then for all practical purposes, it contains AC-3 packs. To explain, the DVD specification states that AC-3 is mandatory for multichannel audio content in North America. By contrast, European (or region two) content initially mandated that MPEG-2 audio be the default multichannel audio standard. Recently, the Region Two Specification was modified to require either AC-3 or MPEG-2 audio for multichannel content. Since every other region in the world requires AC-3, it is likely that AC-3 will become the dominant format in Europe also.

Besides video and audio, VOB files also support subpicture packs. In DVD terminology, a subpicture is a Run Length-compressed bitmap. Each bitmap has a palette of 16 colors, four of which can be active at once. Up to 32 subpicture streams can exist in a given VOB file (usually one stream per language). Unfortunately, since the subpicture palette is so limited, it is difficult to create realistic effects with subpicture alone. As a result, many vendors combine subpicture with high-resolution MPEG-2 video.

The most noticeable use of subpicture is for closed-caption text. Behind the scenes, DVD also uses subpicture in menus. When a menu is displayed, the DVD player modifies the color and contrast of the subpicture for a particular area in the menu, the location appears to be highlighted or selected. As users traverse the menu, the subpicture rectangle is changed so that a selected area moves with them; see Figure 1.

Unlike conventional bitmaps, subpicture data in the stream may be attached to display instructions (or opcodes) that manipulate the image. For instance, there are opcodes that cause the subpicture bitmap to fade or scroll. However, the most interesting opcode is *forcedly start display*. Users often turn off the decoding of a subpicture so that they don't have to view foreign subtitles. When the DVD player encounters forcedly display opcode, the subpicture must always be decoded regardless of user preferences (this is why subpictures in menus will always be displayed even if subpicture decoding is turned off).

Woven among the subpicture, audio, and video are highlight packs. These

highlight packs contain user interface elements called buttons. Buttons are rectangular areas on the screen that monitor user input, and up to 36 buttons can be displayed at any single time. Each button contains associated highlight data structures, and these structures inform the player how to color a button when it is not selected, when it is selected, and when it is chosen (or activated). It also informs the player how long the buttons should remain on the screen and which numerical sequence on the player's remote control can select the button.

DVD Video Title Manager

Besides the VMG, every DVD Video disc contains one or more titles (or movies). These titles are stored in logical containers called "Video Title Sets" (VTS). Like the VMG, there is a strict naming convention for files in a VTS. All files in a VTS are in the form *vts_xx_y* where *xx* is the VTS number (up to a maximum of 99) and *y* is the index within the VTS.

Each VTS has a unique IFO file, vts_xx_y.ifo, and it uses the same data structures as the VMGM: PGCs, programs, and cells. Unlike VMG data structures, VTS data structures often use the exotic capabilities found in PGCs. For instance, title cells can have up to nine different video

Parental Levels

The initial wave of DVD titles had no parental enforcement. The second generation of titles (such as Disney and Universal) have parental controls, but are buggy. For example, Disney content requires players to be at Parental Level Eight (see Examples 1 and 2) before playback can commence, but Level Eight isn't even defined for the United States! Hopefully, as the content matures, parental enforcement will be less problematic.

—L.D.

```
Mov GPRMØ, SPRM13        ; get value of system parental register and copy
                         ;     into a user register #Ø
LT  GPRMØ, 8             ; if  parental level < Max parental level
                         ;     (i.e. 8)
GOTO     Failure         ; then alert user about the failure
```

Example 1: *Poor parental checks in DVD content. Instead of checking for the parental level required by the disc, the content forces the parental level to be at least Level Eight before it will run.*

```
Mov GPRMØ, SPRM13        ; get value of system parental register and copy
                         ;     into a user register #Ø
LT  GPRMØ, #DISC_LEVEL   ; if parental level < the required parental level
                         ;     on the disc
GOTO     Failure         ; then alert user about the failure
```

Example 2: *Correct parental checks in DVD content. In this case, the content verifies that the parental level in the player meets the minimum requirement for the disc, rather than the arbitrary Level Eight in Example 1.*

angles (an angle normally is an alternate camera angle of the presentation). Users can dynamically switch the viewing angle and the player will smoothly transition to a different location in the VOB file based on instructions in the cell.

Title PGCs can also utilize either simple or complex parental controls. Simple parental controls prevent a PGC from playing if users have not authorized its presentation. They will present users with a warning message indicating that the presentation cannot continue until users modify the parental setting for the player.

More sophisticated titles can dynamically select a different PGC depending on the current parental level. They have blocks of PGCs, only one of which will be displayed based on the parental setting. For example, if the player is set for G-rated movies, a nonviolent PGC will be chosen. By contrast, if the player has an R setting, an alternate PGC in the parental block with violent content will be displayed.

Unlike the VMG, a VTS categorizes menus into different topics: chapter, audio, angles, subpicture, and overall title control. These menus contain the same functionality as the VMGM including motion video, background audio, and interactivity via buttons.

Although interactivity has been the most hyped feature in DVD menus, they offer other intriguing options. For instance, DVD menus can be multilingual. To explain, when you create a logical menu screen, it can contain multiple versions of the menu, each in a different language. When the menu is displayed, the DVD player will check the current language system and pick the appropriate menu system for that language. Consequently, you can ship the same disc to different areas of the world, and the DVD players in each region will use the appropriate menu for that language.

Besides internationalization, DVD menus also support the same parental locking features found in title PGCs. You can use these parental controls to display completely different menus depending on the current parental rating system. For example, if the player's parental setting only permits G-rated movies, then the parental block would not show the default PG-13-rated menu, but instead show a special G-rated version that does not give viewers access to the chapters in the movie with sensitive content.

(Many early DVD developers wanted their titles to play on both Windows 95 and dedicated DVD Video machines. Because their programs used the Media Control Interface [which only uses VOB files and ignores the IFO required by DVD-Video], they had to create IFO files for DVD-Video compatibility. Unfortunately, they failed to follow the naming conventions for these files and were bitterly disappointed when they discovered that the content was unusable for DVD Video.)

The DVD specification also defines the minimum set of interactive functions (or operations) a player must provide to the user. Since these capabilities are found in Annex J of the specification, they are often referred to as "Annex J functions." These commands can be divided into the following categories: user interaction via

Region Management

Many users have manually modified their DVD player to ignore region management. This process is usually accomplished by setting all the bits (or enabling all regions) in the player's internal region register. This hack initially allowed players to play content from any region of the world. However, content creators have embedded instructions in the DVD discs to validate that the region code of the player is legitimate. If the content detects that the player has an invalid region code, it will refuse to play the movie (see Example 3).

VMG is also the source of a controversial field — region control. To explain, the DVD specification places artificial limitations on where the disc may be played. If the region code for the player does not match the region code in the content, then the DVD specification will not let the player present the disc — even though the player can decode the content.

—L.D.

```
Mov GPRM0, SPRM20       ; get the region code of the player
NE  GPRM0, 1            ; If region code is not exactly ONE
GOTO    Failure         ; then either this is the wrong player or
                        ;       the user hacked it.
```

Example 3: *Region code checks in content. This sample verifies that the DVD player running the content can play Region One — and only Region One — content by ensuring that only one bit in the region control register is set. If multiple regions are enabled, it will fail.*

buttons, stream controls, random access to presentations, and menu manipulation.

The specification offers commands to navigate through buttons (*UpperButtonSelect()*, *LowerButtonSelect()*, *LeftButtonSelect()*, and *RightButtonSelect()*). Once you've decided on a button, you can use *ButtonSelect()* or *ButtonActivate()* to make a selection.

While a title is playing, you can change the viewing angle via the *Angle_Change()* method. *Audio_Stream_Change()* and *Subpicture_Stream_Change()* let you change audio and subpicture streams (or languages). If you're in a still condition (such as a pause between slides in a slide show), *Still_Off()* causes normal playback to resume.

There are a number of methods that enable random access to content. If you wish to search through the title, you can search via time (*Time_Search()* or *Time_Play()*), by chapter (*Chapter_Play()* and *Chapter_Search()*), or by title (*Title_Play()*).

The *MenuCall()* function lets you display a menu. It has one parameter that dictates which type of menu is displayed (Chapter, Audio, Subpicture, or Title). There are also methods to select Subpicture or Audio streams (*Subpicture_Stream_Change()* and *Audio_Stream_Change()*, respectively), modify parental settings (*Parental_Level_Select()*), and change angles (*Angle_Change()*).

Although Annex J defines the minimum set of interactive functions a DVD player must provide, it is legal, and in some cases, necessary to provide additional functionality for a specific platform. For instance, Microsoft's DirectShow for Win32 (a standard interface and the software drivers required for writing Windows-based DVD applications; see http://www.microsoft.com/directx/) provides enhancements that are specific to the computer environment and not addressed in the specification (see Listings One and Two; listings begin on page 96). It provides methods to process mouse input, finer control of the presentation, and support for asynchronous DVD events. Listing Two illustrates how you process DVD-related DirectShow events.

Conclusion

Unlike VHS, DVD is not simply a linear medium. It was designed to unite computer and consumer electronics users by offering high-quality video, multichannel audio, interactive functions, and a format that can adapt to future technologies. Furthermore, the DVD specification is hardware independent, so your content can run on a wide variety of devices. Once you begin to develop with DVD, you'll never again want to return to today's space constrained, postage-stamp-size multimedia world.

> **For More Information**
>
> Robert's DVD Info:
> http://www.unik.no/~robert/hifi/dvd/
>
> Kilroy's DVD FAQs:
> http://www.CD-info.com/CDIC/Technology/DVD/dvd-faq.html
>
> Chad Fogg's Technical Notes:
> http://www.mpeg.org/~tristan/MPEG/DVD/
>
> DVD-Video Production Guidebook:
> http://www.nbdig.com/html/dvdmain.htm
>
> Quantel Digital Fact Book:
> http://www.quantel.com/dfb/
>
> Sonic DVD Primer:
> http://www.sonic.com/html/dvd/PDF/primer.pdf

DDJ

(Listings begin on page 96.)

68HC05-Based Peripheral Devices: Part II

The keyboard interface as a power supply and communications link

Derrick B. Forte
and Hai T. Nguyen

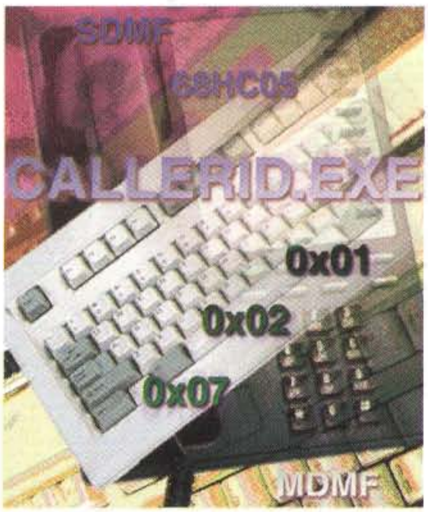
Although the IBM AT's keyboard port is the primary means of user input, you can also use it as the power supply and communications link for small low-power computer peripherals. In this two-part article, we present a system model around which such peripherals can be designed. The application we present is a Caller ID peripheral device based on the Motorola MC68HC(7)05P9 microcontroller. This device is capable of receiving Caller ID transmissions and displaying the received data on an AT-compatible computer. Last month, we focused on the Caller ID protocol and hardware design issues. This month, we complete our discussion of the hardware and zero in on the software.

The authors are engineers at Motorola, and can be contacted at r20367@email.sps.mot.com.

The Caller ID Data-Acquisition Block

The Caller ID data-acquisition block performs two functions within the application's system design:

- Provides an electrical interface to the telephone line.
- Demodulates and validates the Caller ID analog signal and converts it to a digital bit stream.



Though many Caller ID designs implement these functions with discrete analog circuitry, we selected a more integrated solution for this application—Motorola's MC145447 Calling Line Identification Receiver with Ring Detector. This device provides the needed interface to the telephone line, demodulating the BFSK asynchronous data signal, and outputting a digital stream. The design of this block was largely taken from the application note section of the technical data sheet for the MC145447. The device also has a number of signal validation and power saving features that are useful for Caller ID designs for which low power-consumption is an issue. Since this application is powered by the host computer's keyboard interface, it does not use any of the MC145447's power saving modes.

The MC145447's interface to the telephone line's twisted pair can be divided into two types of signals: Caller ID data acquisition signals and ring detection and validation signals. The ring detection and validation signals serve to detect the presence of a valid ring signal on the twisted pair and participate in bringing the device out of power-down mode. There are four signals that comprise the ring detection and validation portion of the interface. Three of the signals—Ring Detect IN 1 (RDI1), Ring Detect IN 2 (RDI2), and /Ring Time (/RT)— are inputs. There is also one output—/Ring Detect Out (/RDO)—which is asserted when a valid power ring is detected on the telephone line twisted pair. The /RT pin works in conjunction with the RDI1 pin to generate internal signals that are part of the

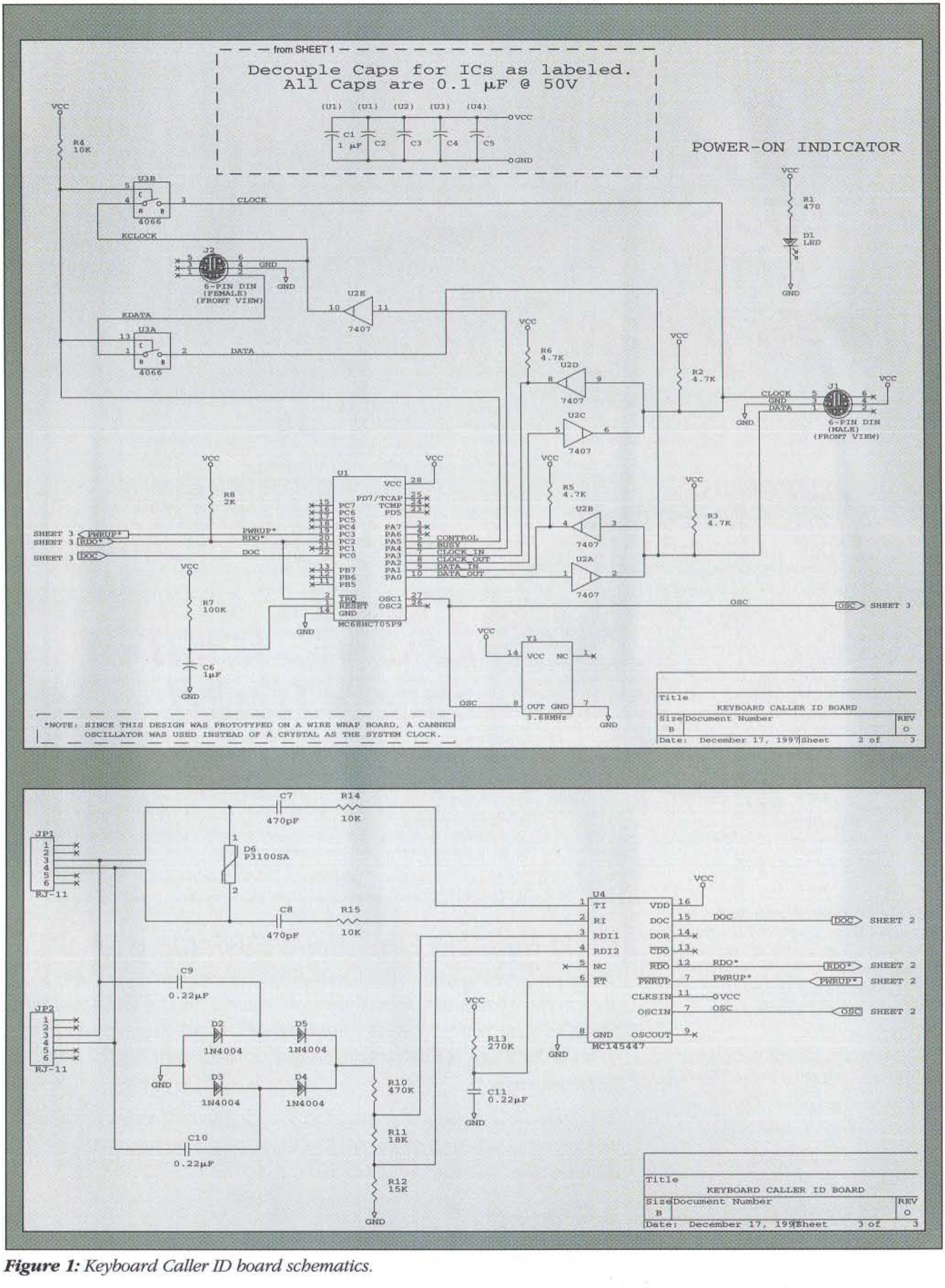


Figure 1: *Keyboard Caller ID board schematics.*

device's power-up circuitry. To conserve power, the MC145447's power-up circuitry applies power to different sections of the device as they are needed.

In the power-up sequence, the /RT and RDI1 signals are used to activate power to the Ring Analysis section of the device. This section determines whether a valid ring signal is present on the twisted pair. As the schematics in Figure 1 shows, the voltage at the RDI1 pin is provided by resistor R10, which is part of a voltage divider circuit comprised of resistors R10, R11, and R12. (More complete schematics are available electronically in .EPS format; see "Resource Center," page 3.) The resistor network divides an AC coupled, rectified version of the voltage present between the tip and ring sides of the twisted pair into voltages that are sampled by the RDI1 and RD2 pins. The value of R10 is chosen such that if a voltage of 40Vrms or more is present on the twisted pair, which indicates that a power ring might be taking place, the RDI1 pin and its associated circuitry will turn power on to the Ring Analysis circuitry. The /RT is connected to a RC combination that holds the pin low during the low periods of a power ring. The RDI2 pin serves as the only input to the Ring Analysis section. The signal at this pin is provided by resistor R12 of the divider network. The duty cycle of this signal is used to validate the presence of a power ring. In the event that a power ring is detected, the Ring Analysis circuit asserts the /RDO pin.

The data-acquisition signals on the MC145447 consists of a Tip input (TI) and Ring input (RI) pin. The TI is AC coupled to the tip side of the telephone line's twisted pair through capacitor C7. The RI signal is AC coupled to the Ring side of the twisted pair through capacitor C8. The signal that is presented to these two pins is demodulated and converted into the digital stream that is output by the device.

In our application, the MC145447's interface with the system's microcontroller consists of three pins — the Data Out Cooked (DOC) pin, the /Ring Detect Out (/RDO) pin, and the /Power Up (/PWRUP) pin. The MC145447 outputs a digital stream on two pins, which are the Data Out Cooked (DOC) pin and the Data Out Raw (DOR) pin.

The DOR pin outputs the entire data stream demodulated by the device starting with the Channel Seizure and Mark Signals and ending with the checksum byte at the end of a transmission. The DOC pin, on the other hand, outputs data after a transmission passes an internal data validation process and does not output the Channel Seizure and Mark Signals. Data is captured by the MC68HC(7)05P9 by connecting DOC to pin PC3 on the MC68HC(7)05P9, which is configured as an input.

The /RDO pin is connected to pin PC2 of the MCU, which is configured as an input. As stated earlier, the /RDO pin is asserted when a valid power ring is detected on the twisted pair. The assertion of the /RDO pin, along with the start of the transmission of data within 0.5–1.5 seconds after the deassertion of /RDO, is used by the MC68HC(7)05P9 to qualify the start of a data stream from the MC145447.

The MC145447 has a requirement that its /PWRUP pin be at a Logic 1 for a minimum of 10µS after VDD reaches its full value. Typically, this requirement is met by delaying the assertion of /PWRUP with a RC circuit. To eliminate the need for these two components, the /PWRUP pin is connected to the MC68HC(7)05P9's PC3 pin, which is configured as an output. This pin asserts /PWRUP after an appropriate delay.

The Keyboard-Interface Block

The main function of the keyboard-interface block is to transmit Caller ID data captured from the MC145447 to an AT-compatible host computer through its keyboard interface. Pins PA0 and PA1 of the MC68HC(7)05P9 serve as the application's keyboard interface's data signal. PA0 is

configured as an output and is used to transmit data to the keyboard interface. PA1 works in conjunction with PA0 and is configured as an input. This arrangement satisfies the AT keyboard interface requirement that the keyboard interface data line be a bidirectional signal that is capable of both transmitting and receiving data to/from the host. Pins PA2 and PA3 function in a way that is similar to the PA0-PA1 pin pair. PA2 is configured as an output and generates the clock signal required for both keyboard-to-host and host-to-keyboard data transfers, and PA3 is confirmed as an input that reads the level on the clock line. Though the clock signal never functions as an input as does the data line, the AT keyboard interface protocol requires that its level be monitored in the event that the host wishes to transmit data to the keyboard. Since PA0 and PA2 are not open-collector outputs, they cannot be directly connected to the data and clock signals of the keyboard and keyboard interface. Therefore, a 7407 open-collector buffer serves as the interface between the MCU's keyboard interface signals and those of keyboard and the keyboard interface.

The design of the keyboard interface does not allow the keyboard to be connected to the interface while another device is transmitting to it. Therefore, the Caller ID device must disconnect the keyboard's clock and data signals from those of the keyboard interface whenever it transmits to the host. Port A pin PA5 is configured as an output and serves as the control signal for the 4066 analog switches that connect or disconnect the keyboard's signals to those of the interface. The number of tasks that a host computer's CPU may need to perform may prevent it from processing a scan code at the time that it is received at the keyboard interface. To prevent user keystrokes from being lost, the keyboard-interface protocol provides for a busy signal that the host sends to the keyboard to prevent it from sending scan codes until the host can process them. The host signals the keyboard that it is busy by holding the clock line low until it can accept new scan codes. While the host is busy, the keyboard stores the scan codes for new keystrokes in its internal buffer. To prevent the loss of any keystrokes that may be generated while the Caller ID device is transmitting to the host, the MC68HC(7)05P9 pulls the clock signal low after it disconnects the keyboard's signals from the interface. Port A pin PA5 is configured as a output and performs this function.

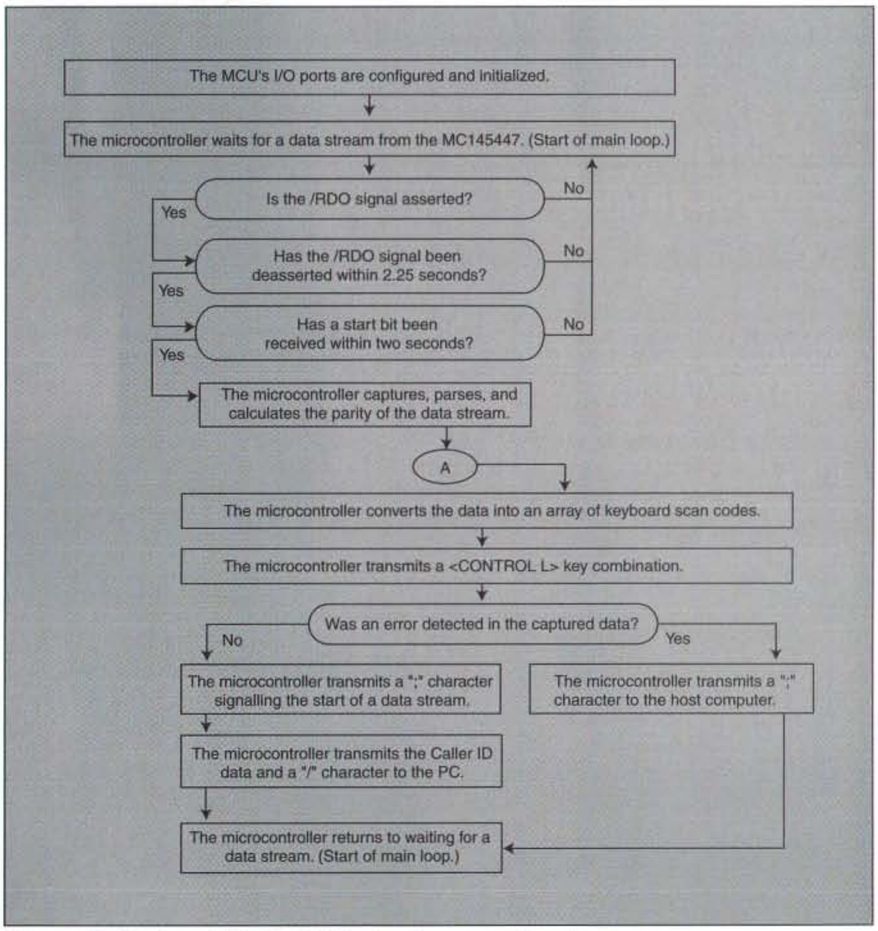


Figure 2: *Keyboard Caller ID device firmware flowchart.*

Keyboard Caller ID Device Software-Design Overview

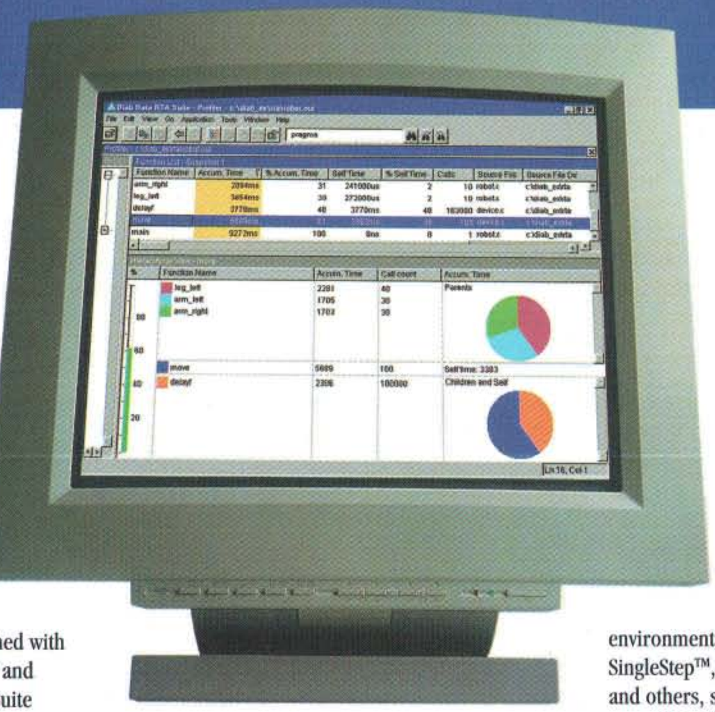
The software design of this application is divided into two parts— the firmware that resides on the MC68HC(7)05P9 and CALLERID.EXE. The firmware's main function is to capture the raw digital data stream generated by the MC145447 and transmit it to the host computer for further processing (source code for the firmware is available electronically; see "Resource Center," page 3). Data is transmitted to the host in the form of keyboard scan codes that are sent through the host's keyboard interface. The host receives the scan codes and interprets them as keystrokes. The sequence of simulated keystrokes is read by CALLERID.EXE, which parses and converts the string back into binary data from which it extracts Caller ID information. CALLERID.EXE (source code for CALLERID.EXE is available electronically) then formats and displays the data in a pop-up dialog box. This division of functionality between the Caller ID device and the host computer allows for the greater portion of processing to be off loaded to the host computer where a larger amount of resources are available. This reduces the functionality of the Caller ID device thus allowing its design to be implemented with a smaller and cheaper microcontroller.

Keyboard Caller ID Device Firmware Design

As Figure 2 illustrates, the Caller ID device's firmware follows this program flow:

1. On reset, the general I/O pins on the MC68HC(7)P9 are configured and initialized to implement the Caller ID device's hardware design.
2. The firmware waits in a loop for the assertion of the MC145447's /RDO signal that is monitored on the MC-68HC(7)05P9's PC2 I/O pin. The assertion of this signal indicates that a power ring has been detected on the twisted pair.
3. If the MC68HC(7)05P9 detects that the MC145447's /RDO pin is deasserted and a start bit on the DOC pin, the conditions are met for the MC-68HC(7)05P9 to begin monitoring for a transmission.
4. The MC145447 transmits the CALLER ID data to the MC68HC(7)05P9 in the form of a raw digital stream on its DOC pin. The MCU reads the data from its PC0 pin.
5. On receiving the data from the MC145447, the MC68HC(7)05P9 parses the stream into individual bytes and checks the data for a parity error. If a parity error has been detected, it is flagged by a global variable, otherwise

SEE WHAT YOU HAVE BEEN MISSING



End The Search!

Looking for development tools to improve program performance, reliability, and memory usage? Look no further! Diab Data's new RTA Suite represents a quantum leap forward in the quest for this exciting new class of tools.

The RTA Suite

Diab Data's RTA (Run-Time Analysis) Suite allows you to capture and analyze your code's run-time behavior, providing you with the critical run-time insight you need to maximize program performance and reliability. Combined with Diab Data's industry-leading D-CC™ and D-C++™ compiler suites, the RTA Suite enables you to develop faster, higher quality code in less time.

Powerful Run-Time Analysis Tools

A Visual Interactive Profiler
for a revealing view of run-time program behavior, including hierarchical "top-down" analysis of parent/children function pairs.

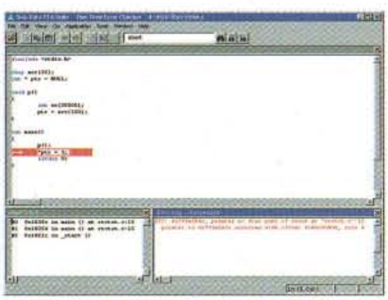
A Run-Time Error Checker
for detecting memory leaks, hard-to-find pointer errors, stack overflows, and many other conditions associated with invalid memory references at run-time.

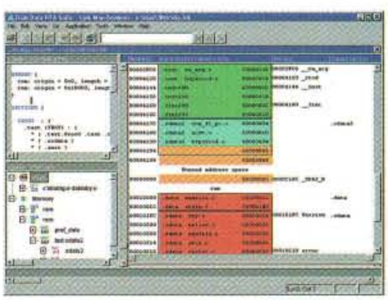
A Visual Link Map Analyzer
for interactive "drag-and-drop" optimization of memory maps.

For Every Developer

The simplicity of the RTA Suite's graphical user interface makes it useful for every developer on your team. Now every developer can benefit from application specific optimizations and the ability to enhance code quality before Q.A. What's more, the RTA Suite is integrated with leading development environments such as ISI's pRISM+™, SDS' SingleStep™, Enea OSE Systems' Illuminator™, and others, so you can access the RTA Suite from familiar tools.

...With The RTA Suite





Diab Data: The Expert's Choice

Diab Data is the compiler leader for embedded PowerPC and Motorola RISC applications. So when you are ready to take your PowerPC, 68K/CPU32, ColdFire, or M•CORE project to the next level, don't fumble around in the dark. Call on the experts.

And see what you have been missing with the RTA Suite!

MOTOROLA
EMBEDDED
PLATINUM DEVELOPER

DIAB DATA
Defining Compiler Performance

Diab Data, Inc., Tel: 650.571.1700, Fax: 650.571.9068. Diab Data GmbH, Tel: +49 (0)89.9393.1191, Fax: +49 (0)89.930.5184

Browse Us www.ddi.com Email Us info@ddi.com

(continued from page 76)

the data is converted into an array of AT keyboard scan codes for transmission to the host computer.

6. The application transmits a <CONTROL L> keystroke sequence as a series of scan codes. This interrupts the application that currently has the focus in Windows 95, activates CALLERID.EXE, and gives it the focus.
7. If a parity error was not detected during the reception of the CALLER ID data, the scan code array that represents the received data is transmitted to the host computer, otherwise an error code is sent.
8. The firmware returns to monitoring the twisted pair for a new Caller ID transmission.

The firmware's functions can be divided into three types of routines:

- Device initialization routines.
- Caller ID data-acquisition routines.
- Keyboard interface routines.

The device initialization routines configure and initialize the MC68HC(7)05P9's I/O pins to implement the application's hardware blocks. As mentioned earlier, Port A I/O pins PA0–PA5 are configured to implement the keyboard interface block, while three Port C pins, PC0, PCG, and PC3, serve as the MC68HC(7)05P9's interface to the MC145447. All remaining general-purpose I/O pins are configured as outputs to eliminate the need for pull-up resistors on them. The data acquisition routines of the firmware consists of the sampling and time delay routines that capture data from the MC145447's DOC line. The MC68HC(7)05P9 samples the data stream at its PC3 pin and parses it into individual bytes. The fact that each piece of Caller ID data begins with a start bit and ends with a stop bit, makes it easy to delineate between individual bytes. The time delay functions used for data acquisition routines are not only used to sample the bits within a byte but must also allow for the inter-character delays that the Interface allows.

The keyboard-interface firmware mainly consists of a transmission routine and its accompanying time delay functions. The keyboard interface's transmit function has within it a call to a routine that is capable of receiving host computer commands. If the host computer detects an error in the data that was sent to it by the keyboard, the host will hold the data low after bad transmission. The host will then send a Resend command (0xFE) to the keyboard requesting a retransmission of the data. Therefore, the Caller ID device must have a receive routine in the event that an error occurs.

For this application, the number of retransmission attempts was arbitrarily set at 1. Therefore, if an error occurs when the device sends a byte to the host, the device will capture the host's resend command and attempt a retransmission of the data. If the retransmission fails, the device will reconnect the keyboard's clock and data signals to those of the host and return to monitoring the telephone line. To transmit data to the host, the transmission routine toggles PA0, which is the data output signal, and the PA2 pin, which is the clock output signal, in accordance with the timing specifications for keyboard-to-computer data transfers. The host command reception routine reads the data from the PA1 pin and toggles the clock signal in accordance with the timing specifications for computer-to-keyboard data transfers.

CALLERID.EXE Design

CALLERID.EXE's design is divided into two parts—CALLERID.EXE (the executable program) and CALLDLL.DLL (the DLL containing the global hook function). Both modules were compiled with Microsoft Visual C++ Version 2.0. CALLDLL.DLL's code consists of a function to install the keyboard hook function and the hook function itself. In the code's call to the Windows API's *SetWindowsHookEx* function, the *idhook* parameter is set to WH_KEYBOARD, which is a predefined value that configures the hook function to handle keyboard events. This code is placed in a DLL because Windows 95 requires that global

hook functions reside in a DLL. The keyboard-hook function in this application must be global in scope so that CALLERID.EXE can be invoked regardless of what application may currently have the focus in Windows 95. The only limitation with CALLERID.EXE is that it will not be invoked if the current window with the focus is a DOS window.

The main function of the executable is to receive the Caller ID data from the Caller ID device, format it, and display it in a dialog box on the PC's monitor. As Figure 3 illustrates, the program flow of the executable is as follows:

1. CALLERID.EXE is invoked immediately after Windows 95 boots up. The main window of the CALLERID application is initialized to come up in the hidden state. This causes CALLERID.EXE to begin executing in the background of Windows 95.
2. CALLERID.EXE accesses CALLERID.DLL and installs the keyboard hook function into the Windows 95 stream. The hook function now examines each keystroke that is entered by the user for the <CONTROL L> hotkey sequence.
3. On detecting a <CONTROL L> key combination, the keyboard hook function calls the Windows API *FindWindow()*, function to locate the application's hidden main window. The Windows *ShowWindow()* function is then called to activate CALLERID.EXE's main window and give it the focus in Windows 95.
4. CALLERID.EXE displays a pop-up dialog box on the monitor displaying the text: "Receiving Data...".
5. The application waits for a keystroke from the Caller ID device.
6. If CALLERID.EXE receives a ";" character from the Caller ID device, the device has detected a parity error in the Caller ID data received from the telephone line. The CALLERID.EXE will then display "Line Error" in the dialog box: Otherwise, it acquires the full stream of Caller ID data from the device.
7. C-string manipulation functions are used to parse the string into the two character segments that represent each byte of Caller ID data. C string conversion functions are then used to convert each ASCII segment into the original binary data that was captured on the Caller ID device.
8. CALLERID.EXE formats the binary data so that it can be displayed in the dialog box. CALLERID.EXE will format data according to whether the Caller ID data received is in the SDMF or MDMF format.
9. The Caller ID information is displayed in the dialog box. The dialog box remains displayed until users press one of the box's OK or Deactivate buttons.
10. The dialog box is hidden again if the user presses the OK button. CALLERID.EXE then returns to waiting for a hot key sequence. If the Deactivate button is pressed, CALLER.EXE will be deactivated and will no longer function until Windows 95 is reset.

Keyboard Caller ID Device Operating Instructions

To use the Keyboard Caller ID system we've presented here:

1. Copy CALLERID.EXE to the hard drive and directory of your choice. A suggested path might be: C:\CALLERID\.
2. Copy CALLDLL.DLL to the C:\WINDOWS\SYSTEM\ directory.
3. Add CALLERID.EXE to the Windows 95 Start Menu.
4. Disconnect the keyboard's connector from the host computer's keyboard port.
5. Connect the Keyboard Caller ID device to the host computer's keyboard interface.
6. Connect the keyboard's connector to the receptacle for it on the Keyboard Caller ID device.
7. Connect the telephone line to one of the R-J11 connectors on the Keyboard Caller ID device.
8. Connect a telephone extension line between the Keyboard Caller ID's second

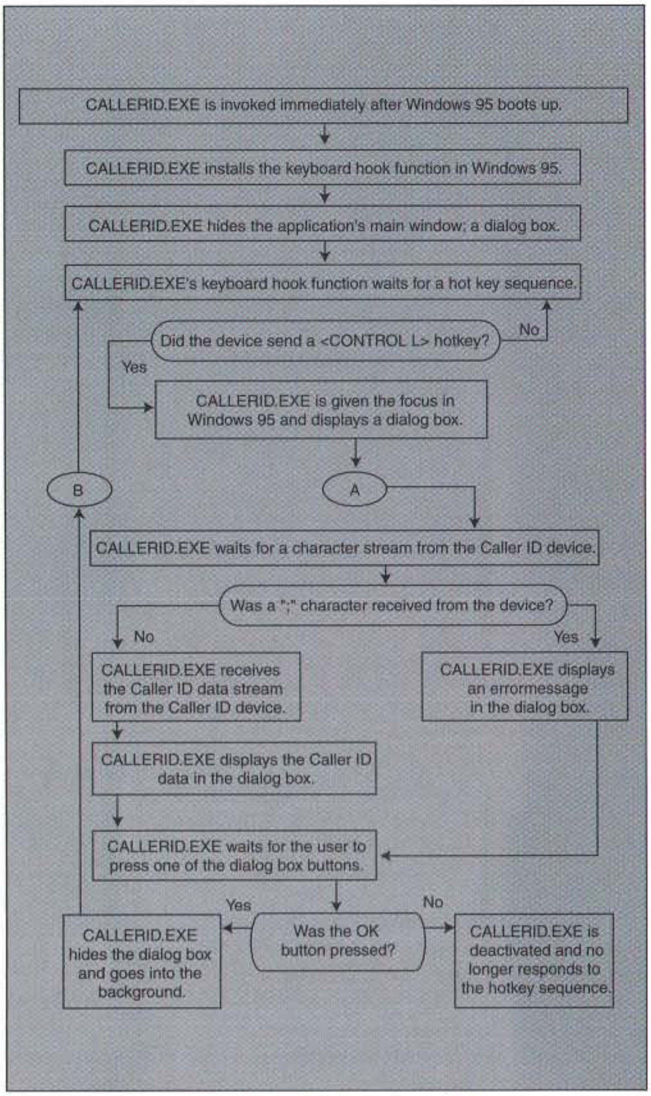


Figure 3: *CALLERID.EXE program flowchart.*

R-J11 connector and your telephone. This completes the hardware installation of the Keyboard Caller ID device.

9. Shut down and restart Windows 95.
10. Caller ID should now activate. Caller ID will display a dialog box with Caller ID information every time a valid transmission is received. To deactivate the program, press the Deactivate button in the dialog box.

References

Holzner, Steve. *Advanced Visual C++ 4*. M&T Books, 1996.

Konzak, Gary J. *PC Keyboard Design*. Second Edition, Annabooks, 1993.

Motorola MC68HC705P9 Technical Data Specification.

Motorola MC145447 Technical Data Specification.

Messmer, Hans-Peter. *The Indispensable PC Hardware Book: Your Questions Answered*. Addison-Wesley, 1994.

LSSGR: Voiceband Data Transmission Interface Section 6.6, GR-30-CORE, Issue 1, Bellcore. December 1994.

"Message Type Values for SDMF and MDMF." *Digest of Technical Information* Bellcore. May 1996.

DDJ

Rendering XML Documents Using XSL

Keeping content and format separate

Sean McGrath

Central to the eXtensible Markup Language (XML) philosophy is that the structure and content of information should be captured without concern for how the information will be rendered on a computer display, paper, voice synthesis, and others. Responsibility for rendering XML has been delegated to a sister standard known as eXtensible Style Language (XSL). (For more information on XML, see my article "XML Programming in Python," *DDJ*, February 1998.)

Like XML, XSL is a World Wide Web Consortium (W3C) initiative. In August of 1997, a draft proposal for XSL was made available as a discussion document by the W3C (http://www.w3.org/TR/NOTE-XSL.html). Although, the working draft for XSL is just that, a number of XSL applications have already appeared. In particular, Microsoft has released MSXSL, a "technology preview" implementation that is freely available at http://www.microsoft.com/xml/. In this article, I will present an overview of XSL and illustrate how it can be used with MSXSL.

Sean, chief technical officer and cofounder of Digitome Electronic Publishing (http://www.digitome.com/) is a member of the World Wide Web Consortium's XML Special Interest Group. He is also the author of ParseMe.1st: SGML for Software Developers *(Prentice-Hall, 1997) and* XML By Example: Building E-commerce Applications *(Prentice-Hall, 1998). Sean can be reached at sean@digitome.com.*

The XSL Philosophy

As Figure 1 illustrates, the XSL philosophy can be summed up as "late binding of presentation semantics." In simple English, the idea is that information about how a document should look when rendered (presentation semantics) is separated from the document content and housed in a stylesheet. The process of creating a rendition of the content happens late — preferably right at the point that someone wants to view it (hence, late binding).

This late binding approach has some significant benefits:

- The look and feel of a document (or thousands of documents) can easily be changed simply by changing the stylesheet.
- Multiple renditions of the same content can be created from a single source. These renditions can include different output notations such as RTF, HTML, or Postscript. They can involve rearrangements of the content, creating multiple views of the information.
- The information content is "future proofed." Creating a new rendition to a new notation (or a notation yet to be invented), is a matter of applying the necessary stylesheet.
- Keeping the content free of rendering information makes it easier to process the content. That is, searching, harvesting, or rearranging the content can be performed without worrying about how the formatting information is intermingled with the content.

There are a number of core concepts that are central to XSL, including:

Flow Objects. In XSL, the process of transforming an XML document into a notation such as RTF, HTML, or Postscript, is expressed in terms of the construction of flow objects, which are

pages, columns, paragraphs, table cells, and so on.

Platform-Independent Flow Objects. XSL specifies a set of standard flow objects such as paragraph, page sequence, table, and the like. Using these platform-independent flow objects lets you create multiple output notations with a single XSL stylesheet. The type of notations that can be created is limited only by the back-end notations supported by the XSL processor. Strong candidates for XSL back ends include RTF, FrameMaker MIF, and TeX.

HTML-Specific Flow Objects. To facilitate the use of XSL stylesheets to generate HTML, XSL provides a set of HTML-specific flow objects. Given the vast amount of HTML-aware software in existence, it makes sense to use this software, while simultaneously retaining the advantages of XML over HTML as a data representation.

Construction Rules. Flow-object construction in XSL is controlled by rules in the XSL stylesheet. These rules specify what flow objects are to be created and what they should contain. Flow objects can be thought of as containers for document content and/or other flow objects creating a tree-like hierarchy known as a "flow-object tree." Flow-object construction rules take the form of a pattern and action. The pattern part specifies the conditions under which the rule triggers. The action part specifies what flow objects to construct.

Characteristics. Flow objects can have associated characteristics that differ depending on the type of flow object being constructed. A paragraph flow object, for example, might have margin and tab characteristics. A table cell might have border and spanning characteristics. The characteristics to be applied to flow objects can be controlled in the XSL stylesheet by means of style rules. Style rules take the same general form as construction rules, and consist of pattern and action components.

Scripting. No stylesheet language that provides a fixed set of rendering capabilities can provide all the processing power needed. There comes a point where a "Turing Complete" programming language is the best way to get the job done. The XSL draft specifies ECMAScript (a standardized version of JavaScript—ECMA 262) as a built-in scripting language. A number of mechanisms are provided in XSL for escaping to ECMAScript to perform calculations, define functions, and so on.

Introducing MSXSL

MSXSL is Microsoft's technology preview implementation of the XSL draft specification. Don't confuse it with MSXML, which is Microsoft's implementation of an XML parser. Indeed, MSXSL uses MSXML to parse and load XSL stylesheets.

MSXSL focuses on creating HTML from XML and, for the time being, only supports HTML flow objects. The simplest way to use MSXSL is via the provided command-line utility that takes the input XML file (-i), input stylesheet file (-s), and output HTML file (-o). For example, the command *C>msxsl -i foo.xml -s foo.xsl -o foo.htm* processes the foo.xml file with respect to the foo.xsl stylesheet specification, then generates the foo.htm output file.

MSXSL is Microsoft's technology preview implementation of the XSL draft specification

To illustrate how to use XSL and MSXSL, I'll return to the XML document (see Figure 2) presented in my February 1998 article.

Sample #1: Getting Started

Listing One(a) (listings begin on page 97) creates a simple stylesheet to convert the XML document in Figure 2 to HTML. Some things to note about this stylesheet:

- It is an XML document and uses a set of element types—*xsl, rule, root,* and so on—defined by the XSL language. The tags for these elements appear in lowercase.
- It signifies the creation of HTML flow objects by using HTML tags—BODY, TITLE, and so on. These tags appear in uppercase.
- It consists of a single flow-object construction rule. The pattern that triggers the rule is the root element (*<root/>*).
- The children element (*<children/>*) tells the XSL processor that all the children of the element that triggered the construction rule should be processed, and the results of processing these elements should be inserted into the output flow-object tree.

Listing One(b) is the result of processing the XML document with this stylesheet. While it's hardly the world's most exciting HTML file, there are some important things to note:

- All the data content of the XML document (the content of the *Maker* and *Color* elements) has found its way into the output document. The default in XSL is that the content of elements that do not trigger construction rules simply flows over to the output document at the point where it is encountered.
- The attribute values (*Price, Units,* and *Type*) do not appear in the output document. This is also the result of the default behavior of XSL.

Sample #2: Rudimentary Formatting

Listing Two(a) adds a few more construction rules to create slightly more pleasing HTML output, while Listing Two(b) presents the result of processing the XML document with this stylesheet. Things to note about this stylesheet and the resultant HTML include:

- The pattern part of the flow-object construction rules use the *target-element* element, which can be used in a variety of ways to specify context-sensitive rules. Here, I used the simplest form in which the *target-element* is an empty element (denoted by the slash in *<target-element/>*).
- The same construction rule can be triggered for multiple element types by specifying multiple empty *target-element* elements. I've used this to cause the same rule to trigger on *Condition* and *Color* elements. The paragraph generated for the *Condition* element is empty because attribute values are not, by default, included in the output document.
- The stylesheet is an XML document and, thus, must be well-formed XML. This is why the HR flow object uses XML syntax to indicate it is an empty element (*<HR/>*).

```
C>msxsl -i cfs.xml -s cfs4.xsl -o cfs4.htm
Error in style sheet 'cfs4.xsl'
ParseException: Expecting name instead of '1'
Location: file:///C:/DOBBS/cfs4.xsl (9,17)
Context: <xsl><rule><HTML><BODY><TABLE>
```

***Example 1:** MSXSL parsing error.*

Sample #3: Accessing Attribute Values

The *Car* elements in Figure 2 have attributes for price and currency information. These can be accessed in XSL by escaping to the ECMAScript scripting language. XSL provides an *eval* element that can be used to house script code. In Listing Three(a), the rule is modified to let *Car* elements access the attribute information. Listing Three(b) is the HTML from this modified stylesheet.

The CDATA section in the *eval* element is an XML construct that shields text from the attentions of the XML parser. The CDATA section begins with the "<![CDATA[" string and ends at the "]]" string. It is a good idea to use CDATA sections to shield script code, since characters such as "<" and "&" can have special meanings to an XML parser.

Sample #4: Creating a Table

Listing Four(a) is a stylesheet creating a simple HTML table layout of "car for sale" information. Listing Four(b) is the result of applying this stylesheet to the XML file.

- All attribute values for the HTML flow objects must be quoted to make the stylesheet well-formed XML. The correct way to specify a table with a border is <TABLE BORDER="1">. The syntax <TABLE BORDER=1>, which is HTML valid, generates a parsing error in MSXSL; see Example 1.
- The first cell in e
the relative num
and so on. This r
ly generated by t
tion. The built-in
tomatic numberin
stylesheets.

Figure 3 shows w
file looks like in Int

Sample #5: Rearr

With XSL, it is possil
the order in which
document are proce
ument content to
ranged prior to crea
ing Five(a), a table
created; Listing Five
applying this stylesl

Only the *Maker*
peared in the outpu
select-element elem
the *Maker* children
processed. By defa
element looks at th
rent element to fin
possible to arrange

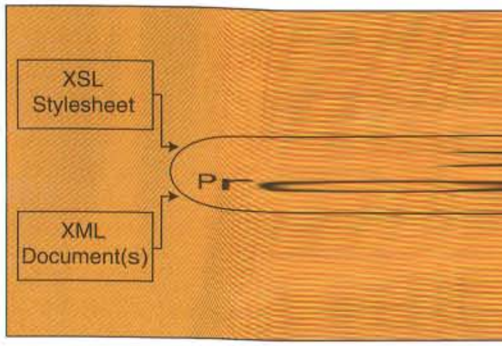


***Figure 1:** The XSL*

```
C>type cars.xml
<?xml version = "1.0"?
<CarsForSale>
    <Car Price = "100
    <Maker>Toyota</M
    <Condition Type =
    <Color>Red</Colo
    </Car>
    <Car Price = "200
    <Maker>Ford</Ma
    <Condition Type =
    <Color>White</Co
    </Car>
</CarsForSale>
```

***Figure 2:** Typical*

***Figure 3:** Typical g*
viewed using Inter

-XSL.html

nl/

tech.com/xsl/xsl-list/

g.ed.ac.uk/~ht/xslj.html

ytech.com/dsssl/dssslist/

rs, which should process and directly.

DDJ

gs begin on page 97.)

Rendering XML Documents Using XSL

Keeping content and format separate

Sean McGrath

Central to the eXtensible Markup Language (XML) philosophy is that the structure and content of information should be captured without concern for how the information will be rendered on a computer display, paper, voice synthesis, and others. Responsibility for rendering XML has been delegated to a sister standard known as eXtensible Style Language (XSL). (For more information on XML, see my article "XML Programming in Python," *DDJ*, February 1998.)

Like XML, XSL is a World Wide Web Consortium (W3C) initiative. In August of 1997, a draft proposal for XSL was made available as a discussion document by the W3C (http://www.w3.org/TR/NOTE-XSL.html). Although, the working draft for XSL is just that, a number of XSL applications have already appeared. In particular, Microsoft has released MSXSL, a "technology preview" implementation that is freely available at http://www.microsoft.com/xml/. In this article, I will present an overview of XSL and illustrate how it can be used with MSXSL.

The XSL Philosophy

As Figure 1 illustrates, the XSL philosophy can be summed up as "late binding of presentation semantics." In simple English, the idea is that information about how a document should look when rendered (presentation semantics) is separated from the document content and housed in a stylesheet. The process of creating a rendition of the content happens late — preferably right at the point that someone wants to view it (hence, late binding).

This late binding approach has some significant benefits:

- The look and feel of a document (or thousands of documents) can easily be changed simply by changing the stylesheet.
- Multiple renditions of the same content can be created from a single source. These renditions can include different output notations such as RTF, HTML, or Postscript. They can involve rearrangements of the content, creating multiple views of the information.
- The information content is "future proofed." Creating a new rendition to a new notation (or a notation yet to be invented), is a matter of applying the necessary stylesheet.
- Keeping the content free of rendering information makes it easier to process the content. That is, searching, harvesting, or rearranging the content can be performed without worrying about how the formatting information is intermingled with the content.

There are a number of core concepts that are central to XSL, including:

Flow Objects. In XSL, the process of transforming an XML document into a notation such as RTF, HTML, or Postscript, is expressed in terms of the construction of flow objects, which are

Sean, chief technical officer and cofounder of Digitome Electronic Publishing (http://www.digitome.com/) is a member of the World Wide Web Consortium's XML Special Interest Group. He is also the author of ParseMe.1st: SGML for Software Developers *(Prentice-Hall, 1997) and* XML By Example: Building E-commerce Applications *(Prentice-Hall, 1998). Sean can be reached at sean@digitome.com.*

Sample #3: Accessing Attribute Values

The *Car* elements in Figure 2 have attributes for price and currency information. These can be accessed in XSL by escaping to the ECMAScript scripting language. XSL provides an *eval* element that can be used to house script code. In Listing Three(a), the rule is modified to let *Car* elements access the attribute information. Listing Three(b) is the HTML from this modified stylesheet.

The CDATA section in the *eval* element is an XML construct that shields text from the attentions of the XML parser. The CDATA section begins with the "<![CDATA[" string and ends at the "]]" string. It is a good idea to use CDATA sections to shield script code, since characters such as "<" and "&" can have special meanings to an XML parser.

Sample #4: Creating a Table

Listing Four(a) is a stylesheet creating a simple HTML table layout of "car for sale" information. Listing Four(b) is the result of applying this stylesheet to the XML file.

- All attribute values for the HTML flow objects must be quoted to make the stylesheet well-formed XML. The correct way to specify a table with a border is <TABLE BORDER="1">. The syntax <TABLE BORDER=1>, which is HTML valid, generates a parsing error in MSXSL; see Example 1.
- The first cell in each table row contains the relative number of the *Car*—1, 2, and so on. This number is automatically generated by the *childNumber()* function. The built-in XSL functions for automatic numbering are useful in creating stylesheets.

Figure 3 shows what the generated HTML file looks like in Internet Explorer 4.0.

Sample #5: Rearranging Content

With XSL, it is possible to exert control over the order in which elements in the source document are processed. This allows document content to be selected and rearranged prior to creating the output. In Listing Five(a), a table of car maker names is created; Listing Five(b) shows the result of applying this stylesheet to the XML file.

Only the *Maker* element data has appeared in the output. This is because the *select-element* element indicates that only the *Maker* children of *Car* elements are processed. By default, the *select-element* element looks at the children of the current element to find matches. It is also possible to arrange for *select-element* to

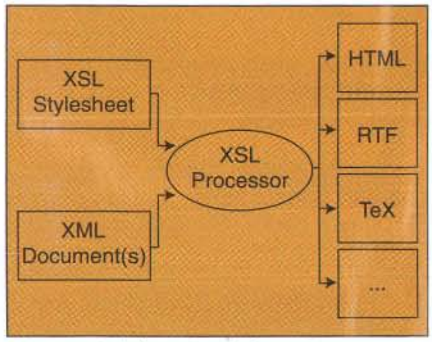


Figure 1: *The XSL philosophy.*

```
C>type cars.xml
<?xml version = "1.0"?>
<CarsForSale>
    <Car Price = "10000" Units = "Dollars">
    <Maker>Toyota</Maker>
    <Condition Type = "Good"/>
    <Color>Red</Color>
    </Car>
    <Car Price = "20000" Units = "Irish Punts">
    <Maker>Ford</Maker>
    <Condition Type = "Good"/>
    <Color>White</Color>
    </Car>
</CarsForSale>
```

Figure 2: *Typical XML document.*

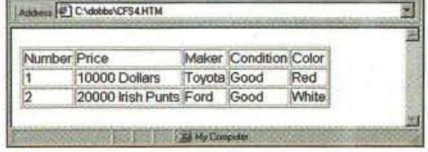


Figure 3: *Typical generated HTML file viewed using Internet Explorer 4.0.*

pages, columns, paragraphs, table cells, and so on.

Platform-Independent Flow Objects. XSL specifies a set of standard flow objects such as paragraph, page sequence, table, and the like. Using these platform-independent flow objects lets you create multiple output notations with a single XSL stylesheet. The type of notations that can be created is limited only by the back-end notations supported by the XSL processor. Strong candidates for XSL back ends include RTF, FrameMaker MIF, and TeX.

HTML-Specific Flow Objects. To facilitate the use of XSL stylesheets to generate HTML, XSL provides a set of HTML-specific flow objects. Given the vast amount of HTML-aware software in existence, it makes sense to use this software, while simultaneously retaining the advantages of XML over HTML as a data representation.

Construction Rules. Flow-object construction in XSL is controlled by rules in the XSL stylesheet. These rules specify what flow objects are to be created and what they should contain. Flow objects can be thought of as containers for document content and/or other flow objects creating a tree-like hierarchy known as a "flow-object tree." Flow-object construction rules take the form of a pattern and action. The pattern part specifies the conditions under which the rule triggers. The action part specifies what flow objects to construct.

Characteristics. Flow objects can have associated characteristics that differ depending on the type of flow object being constructed. A paragraph flow object, for example, might have margin and tab characteristics. A table cell might have border and spanning characteristics. The characteristics to be applied to flow objects can be controlled in the XSL stylesheet by means of style rules. Style rules take the same general form as construction rules, and consist of pattern and action components.

Scripting. No stylesheet language that provides a fixed set of rendering capabilities can provide all the processing power needed. There comes a point where a "Turing Complete" programming language is the best way to get the job done. The XSL draft specifies ECMAScript (a standardized version of JavaScript—ECMA 262) as a built-in scripting language. A number of mechanisms are provided in XSL for escaping to ECMAScript to perform calculations, define functions, and so on.

Introducing MSXSL

MSXSL is Microsoft's technology preview implementation of the XSL draft specification. Don't confuse it with MSXML, which is Microsoft's implementation of an XML parser. Indeed, MSXSL uses MSXML to parse and load XSL stylesheets.

MSXSL focuses on creating HTML from XML and, for the time being, only supports HTML flow objects. The simplest way to use MSXSL is via the provided command-line utility that takes the input XML file (-i), input stylesheet file (-s), and output HTML file (-o). For example, the command *C>msxsl -i foo.xml -s foo.xsl -o foo.htm* processes the foo.xml file with respect to the foo.xsl stylesheet specification, then generates the foo.htm output file.

MSXSL is Microsoft's technology preview implementation of the XSL draft specification

To illustrate how to use XSL and MSXSL, I'll return to the XML document (see Figure 2) presented in my February 1998 article.

Sample #1: Getting Started

Listing One(a) (listings begin on page 97) creates a simple stylesheet to convert the XML document in Figure 2 to HTML. Some things to note about this stylesheet:

- It is an XML document and uses a set of element types—*xsl, rule, root,* and so on—defined by the XSL language. The tags for these elements appear in lowercase.
- It signifies the creation of HTML flow objects by using HTML tags—BODY, TITLE, and so on. These tags appear in uppercase.
- It consists of a single flow-object construction rule. The pattern that triggers the rule is the root element (*<root/>*).
- The children element (*<children/>*) tells the XSL processor that all the children of the element that triggered the construction rule should be processed, and the results of processing these elements should be inserted into the output flow-object tree.

Listing One(b) is the result of processing the XML document with this stylesheet. While it's hardly the world's most exciting HTML file, there are some important things to note:

- All the data content of the XML document (the content of the *Maker* and *Color* elements) has found its way into the output document. The default in XSL is that the content of elements that do not trigger construction rules simply flows over to the output document at the point where it is encountered.
- The attribute values (*Price, Units,* and *Type*) do not appear in the output document. This is also the result of the default behavior of XSL.

Sample #2: Rudimentary Formatting

Listing Two(a) adds a few more construction rules to create slightly more pleasing HTML output, while Listing Two(b) presents the result of processing the XML document with this stylesheet. Things to note about this stylesheet and the resultant HTML include:

- The pattern part of the flow-object construction rules use the *target-element* element, which can be used in a variety of ways to specify context-sensitive rules. Here, I used the simplest form in which the *target-element* is an empty element (denoted by the slash in *<target-element/>*).
- The same construction rule can be triggered for multiple element types by specifying multiple empty *target-element* elements. I've used this to cause the same rule to trigger on *Condition* and *Color* elements. The paragraph generated for the *Condition* element is empty because attribute values are not, by default, included in the output document.
- The stylesheet is an XML document and, thus, must be well-formed XML. This is why the HR flow object uses XML syntax to indicate it is an empty element (*<HR/>*).

```
C>msxsl -i cfs.xml -s cfs4.xsl -o cfs4.htm
Error in style sheet 'cfs4.xsl'
ParseException: Expecting name instead of '1'
Location: file:///C:/DOBBS/cfs4.xsl (9,17)
Context: <xsl><rule><HTML><BODY><TABLE>
```

Example 1: *MSXSL parsing error.*

Rendering XML Documents Using XSL

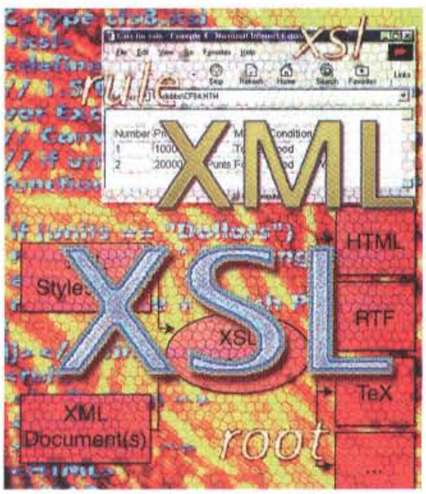
Keeping content and format separate

Sean McGrath

Central to the eXtensible Markup Language (XML) philosophy is that the structure and content of information should be captured without concern for how the information will be rendered on a computer display, paper, voice synthesis, and others. Responsibility for rendering XML has been delegated to a sister standard known as eXtensible Style Language (XSL). (For more information on XML, see my article "XML Programming in Python," *DDJ*, February 1998.)

Like XML, XSL is a World Wide Web Consortium (W3C) initiative. In August of 1997, a draft proposal for XSL was made available as a discussion document by the W3C (http://www.w3.org/TR/NOTE-XSL.html). Although, the working draft for XSL is just that, a number of XSL applications have already appeared. In particular, Microsoft has released MSXSL, a "technology preview" implementation that is freely available at http://www.microsoft.com/xml/. In this article, I will present an overview of XSL and illustrate how it can be used with MSXSL.

Sean, chief technical officer and cofounder of Digitome Electronic Publishing (http://www.digitome.com/) is a member of the World Wide Web Consortium's XML Special Interest Group. He is also the author of ParseMe.1st: SGML for Software Developers *(Prentice-Hall, 1997) and* XML By Example: Building E-commerce Applications *(Prentice-Hall, 1998). Sean can be reached at sean@digitome.com.*



The XSL Philosophy

As Figure 1 illustrates, the XSL philosophy can be summed up as "late binding of presentation semantics." In simple English, the idea is that information about how a document should look when rendered (presentation semantics) is separated from the document content and housed in a stylesheet. The process of creating a rendition of the content happens late — preferably right at the point that someone wants to view it (hence, late binding).

This late binding approach has some significant benefits:

- The look and feel of a document (or thousands of documents) can easily be changed simply by changing the stylesheet.
- Multiple renditions of the same content can be created from a single source. These renditions can include different output notations such as RTF, HTML, or Postscript. They can involve rearrangements of the content, creating multiple views of the information.
- The information content is "future proofed." Creating a new rendition to a new notation (or a notation yet to be invented), is a matter of applying the necessary stylesheet.
- Keeping the content free of rendering information makes it easier to process the content. That is, searching, harvesting, or rearranging the content can be performed without worrying about how the formatting information is intermingled with the content.

There are a number of core concepts that are central to XSL, including:

Flow Objects. In XSL, the process of transforming an XML document into a notation such as RTF, HTML, or Postscript, is expressed in terms of the construction of flow objects, which are

pages, columns, paragraphs, table cells, and so on.

Platform-Independent Flow Objects. XSL specifies a set of standard flow objects such as paragraph, page sequence, table, and the like. Using these platform-independent flow objects lets you create multiple output notations with a single XSL stylesheet. The type of notations that can be created is limited only by the back-end notations supported by the XSL processor. Strong candidates for XSL back ends include RTF, FrameMaker MIF, and TeX.

HTML-Specific Flow Objects. To facilitate the use of XSL stylesheets to generate HTML, XSL provides a set of HTML-specific flow objects. Given the vast amount of HTML-aware software in existence, it makes sense to use this software, while simultaneously retaining the advantages of XML over HTML as a data representation.

Construction Rules. Flow-object construction in XSL is controlled by rules in the XSL stylesheet. These rules specify what flow objects are to be created and what they should contain. Flow objects can be thought of as containers for document content and/or other flow objects creating a tree-like hierarchy known as a "flow-object tree." Flow-object construction rules take the form of a pattern and action. The pattern part specifies the conditions under which the rule triggers. The action part specifies what flow objects to construct.

Characteristics. Flow objects can have associated characteristics that differ depending on the type of flow object being constructed. A paragraph flow object, for example, might have margin and tab characteristics. A table cell might have border and spanning characteristics. The characteristics to be applied to flow objects can be controlled in the XSL stylesheet by means of style rules. Style rules take the same general form as construction rules, and consist of pattern and action components.

Scripting. No stylesheet language that provides a fixed set of rendering capabilities can provide all the processing power needed. There comes a point where a "Turing Complete" programming language is the best way to get the job done. The XSL draft specifies ECMAScript (a standardized version of JavaScript—ECMA 262) as a built-in scripting language. A number of mechanisms are provided in XSL for escaping to ECMAScript to perform calculations, define functions, and so on.

Introducing MSXSL

MSXSL is Microsoft's technology preview implementation of the XSL draft specification. Don't confuse it with MSXML, which is Microsoft's implementation of an XML parser. Indeed, MSXSL uses MSXML to parse and load XSL stylesheets.

MSXSL focuses on creating HTML from XML and, for the time being, only supports HTML flow objects. The simplest way to use MSXSL is via the provided command-line utility that takes the input XML file (-i), input stylesheet file (-s), and output HTML file (-o). For example, the command *C>msxsl -i foo.xml -s foo.xsl -o foo.htm* processes the foo.xml file with respect to the foo.xsl stylesheet specification, then generates the foo.htm output file.

MSXSL is Microsoft's technology preview implementation of the XSL draft specification

To illustrate how to use XSL and MSXSL, I'll return to the XML document (see Figure 2) presented in my February 1998 article.

Sample #1: Getting Started

Listing One(a) (listings begin on page 97) creates a simple stylesheet to convert the XML document in Figure 2 to HTML. Some things to note about this stylesheet:

- It is an XML document and uses a set of element types—*xsl, rule, root,* and so on—defined by the XSL language. The tags for these elements appear in lowercase.
- It signifies the creation of HTML flow objects by using HTML tags—BODY, TITLE, and so on. These tags appear in uppercase.
- It consists of a single flow-object construction rule. The pattern that triggers the rule is the root element (*<root/>*).
- The children element (*<children/>*) tells the XSL processor that all the children of the element that triggered the construction rule should be processed, and the results of processing these elements should be inserted into the output flow-object tree.

Listing One(b) is the result of processing the XML document with this stylesheet. While it's hardly the world's most exciting HTML file, there are some important things to note:

- All the data content of the XML document (the content of the *Maker* and *Color* elements) has found its way into the output document. The default in XSL is that the content of elements that do not trigger construction rules simply flows over to the output document at the point where it is encountered.
- The attribute values (*Price, Units,* and *Type*) do not appear in the output document. This is also the result of the default behavior of XSL.

Sample #2: Rudimentary Formatting

Listing Two(a) adds a few more construction rules to create slightly more pleasing HTML output, while Listing Two(b) presents the result of processing the XML document with this stylesheet. Things to note about this stylesheet and the resultant HTML include:

- The pattern part of the flow-object construction rules use the *target-element* element, which can be used in a variety of ways to specify context-sensitive rules. Here, I used the simplest form in which the *target-element* is an empty element (denoted by the slash in *<target-element/>*).
- The same construction rule can be triggered for multiple element types by specifying multiple empty *target-element* elements. I've used this to cause the same rule to trigger on *Condition* and *Color* elements. The paragraph generated for the *Condition* element is empty because attribute values are not, by default, included in the output document.
- The stylesheet is an XML document and, thus, must be well-formed XML. This is why the HR flow object uses XML syntax to indicate it is an empty element (*<HR/>*).

```
C>msxsl -i cfs.xml -s cfs4.xsl -o cfs4.htm
Error in style sheet 'cfs4.xsl'
ParseException: Expecting name instead of '1'
Location: file:///C:/DOBBS/cfs4.xsl (9,17)
Context: <xsl><rule><HTML><BODY><TABLE>
```

Example 1: *MSXSL parsing error.*

Sample #3: Accessing Attribute Values

The *Car* elements in Figure 2 have attributes for price and currency information. These can be accessed in XSL by escaping to the ECMAScript scripting language. XSL provides an *eval* element that can be used to house script code. In Listing Three(a), the rule is modified to let *Car* elements access the attribute information. Listing Three(b) is the HTML from this modified stylesheet.

The CDATA section in the *eval* element is an XML construct that shields text from the attentions of the XML parser. The CDATA section begins with the "<![CDATA[" string and ends at the "]]" string. It is a good idea to use CDATA sections to shield script code, since characters such as "<" and "&" can have special meanings to an XML parser.

Sample #4: Creating a Table

Listing Four(a) is a stylesheet creating a simple HTML table layout of "car for sale" information. Listing Four(b) is the result of applying this stylesheet to the XML file.

- All attribute values for the HTML flow objects must be quoted to make the stylesheet well-formed XML. The correct way to specify a table with a border is <TABLE BORDER="1">. The syntax <TABLE BORDER=1>, which is HTML valid, generates a parsing error in MSXSL; see Example 1.

- The first cell in eac[illegible] the relative numb[illegible] and so on. This nu[illegible] ly generated by the [illegible] tion. The built-in X[illegible] tomatic numbering [illegible] stylesheets.

Figure 3 shows what [illegible] file looks like in Inter[illegible]

Sample #5: Rearran[illegible]

With XSL, it is possible [illegible] the order in which el[illegible] document are process[illegible] ument content to be [illegible] ranged prior to creatin[illegible] ing Five(a), a table of [illegible] created; Listing Five(b) [illegible] applying this styleshe[illegible]

Only the *Maker* el[illegible] peared in the output. [illegible] *select-element* elemen[illegible] the *Maker* children of [illegible] processed. By default, [illegible] element looks at the [illegible] rent element to find [illegible] possible to arrange fo[illegible]

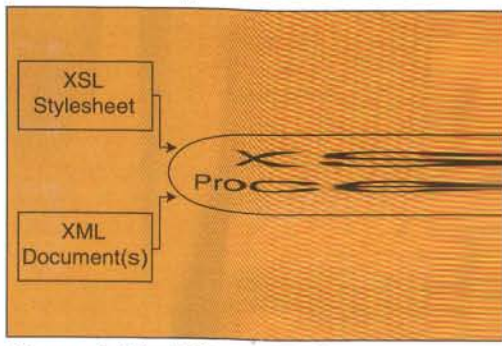


Figure 1: The XSL p[illegible]

```
C>type cars.xml
<?xml version = "1.0"?>
<CarsForSale>
    <Car Price = "10000"
    <Maker>Toyota</Ma
    <Condition Type = "
    <Color>Red</Color>
    </Car>
    <Car Price = "20000"
    <Maker>Ford</Make
    <Condition Type = "
    <Color>White</Color
    </Car>
</CarsForSale>
```

Figure 2: Typical X[illegible]

Figure 3: Typical ge[illegible] *viewed using Interne*[illegible]

Sample #3: Accessing Attribute Values

The *Car* elements in Figure 2 have attributes for price and currency information. These can be accessed in XSL by escaping to the ECMAScript scripting language. XSL provides an *eval* element that can be used to house script code. In Listing Three(a), the rule is modified to let *Car* elements access the attribute information. Listing Three(b) is the HTML from this modified stylesheet.

The CDATA section in the *eval* element is an XML construct that shields text from the attentions of the XML parser. The CDATA section begins with the "<![CDATA[" string and ends at the "]]" string. It is a good idea to use CDATA sections to shield script code, since characters such as "<" and "&" can have special meanings to an XML parser.

Sample #4: Creating a Table

Listing Four(a) is a stylesheet creating a simple HTML table layout of "car for sale" information. Listing Four(b) is the result of applying this stylesheet to the XML file.

- All attribute values for the HTML flow objects must be quoted to make the stylesheet well-formed XML. The correct way to specify a table with a border is <TABLE BORDER="1">. The syntax <TABLE BORDER=1>, which is HTML valid, generates a parsing error in MSXSL; see Example 1.
- The first cell in each table row contains the relative number of the *Car*—1, 2, and so on. This number is automatically generated by the *childNumber()* function. The built-in XSL functions for automatic numbering are useful in creating stylesheets.

Figure 3 shows what the generated HTML file looks like in Internet Explorer 4.0.

Sample #5: Rearranging Content

With XSL, it is possible to exert control over the order in which elements in the source document are processed. This allows document content to be selected and rearranged prior to creating the output. In Listing Five(a), a table of car maker names is created; Listing Five(b) shows the result of applying this stylesheet to the XML file.

Only the *Maker* element data has appeared in the output. This is because the *select-element* element indicates that only the *Maker* children of *Car* elements are processed. By default, the *select-element* element looks at the children of the current element to find matches. It is also possible to arrange for *select-element* to

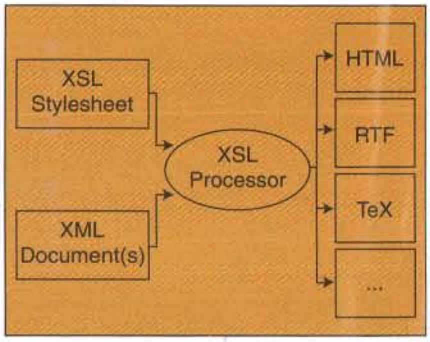


Figure 1: *The XSL philosophy.*

```
C>type cars.xml
<?xml version = "1.0"?>
<CarsForSale>
    <Car Price = "10000" Units = "Dollars">
    <Maker>Toyota</Maker>
    <Condition Type = "Good"/>
    <Color>Red</Color>
    </Car>
    <Car Price = "20000" Units = "Irish Punts">
    <Maker>Ford</Maker>
    <Condition Type = "Good"/>
    <Color>White</Color>
    </Car>
</CarsForSale>
```

Figure 2: *Typical XML document.*

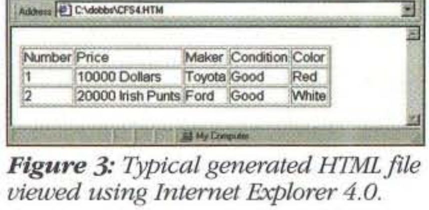


Figure 3: *Typical generated HTML file viewed using Internet Explorer 4.0.*

pages, columns, paragraphs, table cells, and so on.

Platform-Independent Flow Objects. XSL specifies a set of standard flow objects such as paragraph, page sequence, table, and the like. Using these platform-independent flow objects lets you create multiple output notations with a single XSL stylesheet. The type of notations that can be created is limited only by the back-end notations supported by the XSL processor. Strong candidates for XSL back ends include RTF, FrameMaker MIF, and TeX.

HTML-Specific Flow Objects. To facilitate the use of XSL stylesheets to generate HTML, XSL provides a set of HTML-specific flow objects. Given the vast amount of HTML-aware software in existence, it makes sense to use this software, while simultaneously retaining the advantages of XML over HTML as a data representation.

Construction Rules. Flow-object construction in XSL is controlled by rules in the XSL stylesheet. These rules specify what flow objects are to be created and what they should contain. Flow objects can be thought of as containers for document content and/or other flow objects creating a tree-like hierarchy known as a "flow-object tree." Flow-object construction rules take the form of a pattern and action. The pattern part specifies the conditions under which the rule triggers. The action part specifies what flow objects to construct.

Characteristics. Flow objects can have associated characteristics that differ depending on the type of flow object being constructed. A paragraph flow object, for example, might have margin and tab characteristics. A table cell might have border and spanning characteristics. The characteristics to be applied to flow objects can be controlled in the XSL stylesheet by means of style rules. Style rules take the same general form as construction rules, and consist of pattern and action components.

Scripting. No stylesheet language that provides a fixed set of rendering capabilities can provide all the processing power needed. There comes a point where a "Turing Complete" programming language is the best way to get the job done. The XSL draft specifies ECMAScript (a standardized version of JavaScript—ECMA 262) as a built-in scripting language. A number of mechanisms are provided in XSL for escaping to ECMAScript to perform calculations, define functions, and so on.

Introducing MSXSL

MSXSL is Microsoft's technology preview implementation of the XSL draft specification. Don't confuse it with MSXML, which is Microsoft's implementation of an XML parser. Indeed, MSXSL uses MSXML to parse and load XSL stylesheets.

MSXSL focuses on creating HTML from XML and, for the time being, only supports HTML flow objects. The simplest way to use MSXSL is via the provided command-line utility that takes the input XML file (-i), input stylesheet file (-s), and output HTML file (-o). For example, the command *C>msxsl -i foo.xml -s foo.xsl -o foo.htm* processes the foo.xml file with respect to the foo.xsl stylesheet specification, then generates the foo.htm output file.

MSXSL is Microsoft's technology preview implementation of the XSL draft specification

To illustrate how to use XSL and MSXSL, I'll return to the XML document (see Figure 2) presented in my February 1998 article.

Sample #1: Getting Started

Listing One(a) (listings begin on page 97) creates a simple stylesheet to convert the XML document in Figure 2 to HTML. Some things to note about this stylesheet:

- It is an XML document and uses a set of element types—*xsl, rule, root,* and so on—defined by the XSL language. The tags for these elements appear in lowercase.
- It signifies the creation of HTML flow objects by using HTML tags—BODY, TITLE, and so on. These tags appear in uppercase.
- It consists of a single flow-object construction rule. The pattern that triggers the rule is the root element (*<root/>*).
- The children element (*<children/>*) tells the XSL processor that all the children of the element that triggered the construction rule should be processed, and the results of processing these elements should be inserted into the output flow-object tree.

Listing One(b) is the result of processing the XML document with this stylesheet. While it's hardly the world's most exciting HTML file, there are some important things to note:

- All the data content of the XML document (the content of the *Maker* and *Color* elements) has found its way into the output document. The default in XSL is that the content of elements that do not trigger construction rules simply flows over to the output document at the point where it is encountered.
- The attribute values (*Price*, *Units*, and *Type*) do not appear in the output document. This is also the result of the default behavior of XSL.

Sample #2: Rudimentary Formatting

Listing Two(a) adds a few more construction rules to create slightly more pleasing HTML output, while Listing Two(b) presents the result of processing the XML document with this stylesheet. Things to note about this stylesheet and the resultant HTML include:

- The pattern part of the flow-object construction rules use the *target-element* element, which can be used in a variety of ways to specify context-sensitive rules. Here, I used the simplest form in which the *target-element* is an empty element (denoted by the slash in *<target-element/>*).
- The same construction rule can be triggered for multiple element types by specifying multiple empty *target-element* elements. I've used this to cause the same rule to trigger on *Condition* and *Color* elements. The paragraph generated for the *Condition* element is empty because attribute values are not, by default, included in the output document.
- The stylesheet is an XML document and, thus, must be well-formed XML. This is why the HR flow object uses XML syntax to indicate it is an empty element (*<HR/>*).

```
C>msxsl -i cfs.xml -s cfs4.xsl -o cfs4.htm
Error in style sheet 'cfs4.xsl'
ParseException: Expecting name instead of '1'
Location: file:///C:/DOBBS/cfs4.xsl (9,17)
Context: <xsl><rule><HTML><BODY><TABLE>
```

Example 1: *MSXSL parsing error.*